



**AN AUTONOMOUS FRAMEWORK FOR  
REAL-TIME WRONG-WAY DRIVING  
VEHICLE DETECTION FROM CCTV**

**BY**

**PINTUSORN SUTTIPONPISARN**

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT  
OF THE REQUIREMENTS FOR THE DEGREE  
OF MASTER OF ENGINEERING  
(ARTIFICIAL INTELLIGENCE AND INTERNET OF THINGS)  
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY  
THAMMASAT UNIVERSITY  
ACADEMIC YEAR 2021  
COPYRIGHT OF THAMMASAT UNIVERSITY**

THAMMASAT UNIVERSITY  
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THESIS

BY

PINTUSORN SUTTIPONPISARN

ENTITLED

AN AUTONOMOUS FRAMEWORK FOR REAL-TIME WRONG-WAY  
DRIVING VEHICLE DETECTION FROM CCTV

was approved as partial fulfillment of the requirements for  
the degree of Master of Engineering (Artificial Intelligence and Internet of Things)

on May 10, 2022

Chairperson



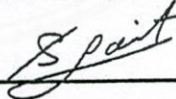
(Chalermopol Charnsripinyo, Ph.D.)

Member and Advisor



(Assistant Professor Sasiporn Usanavasin, Ph.D.)

Member



(Seksan Laitrakun, Ph.D.)

Member



(Associate Professor Hiroki Nakahara, Ph.D.)

Director



(Professor Pruettha Nanakorn, D.Eng.)

Thesis Title	AN AUTONOMOUS FRAMEWORK FOR REAL-TIME WRONG-WAY DRIVING VEHICLE DETECTION FROM CCTV
Author	Pintusorn Suttiponpisarn
Degree	Master of Engineering (Artificial Intelligence and Internet of Things)
Faculty/University	Sirindhorn International Institute of Technology/ Thammasat University
Thesis Advisor	Assistant Professor Sasiporn Usanavasin, Ph.D.
Academic Years	2021

## ABSTRACT

In 2018, around 22,000 people were killed in road accidents in Thailand and more than 80% of road accidents involved motorcycles. There are several reasons for traffic death and one of them is motorcycles driving in the wrong direction. In our research, we proposed an autonomous framework, WrongWay-LVDC, that can detect vehicles that drive in the wrong direction from CCTV videos. The framework is composed of other additional features that allow the detection process to run smoothly and automatically. The features that are included in the framework are road lane detecting, correct driving direction validating, detecting the wrong-way driving vehicles, and evidence image capturing. In total, we have proposed three algorithms and a feature for our main contributions. The three algorithms are, first, the Road Lane Boundary Detection based on the CCTV camera angle (RLB-CCTV) algorithm. Second, is the Majority-Based Correct Direction Detection (MBCDD) algorithm. Third, is the Distance-based Direction Detection (DBDD) algorithm. The additional feature we developed is the Inside Boundary Image (IBI) capturing feature that captures the clearest shot of the wrong-way driving vehicle. Each algorithm and feature will be explained in terms of its methods, result, and discussion in this thesis. As a result, the framework can execute continuously and output a report for vehicles' driving behaviors in each area. The accuracy of our approach is 95.23% as we tested with several CCTV videos using

the DBDD algorithm. Moreover, the framework can be implemented on edge devices with real-time speed where it is proved that the framework is light-weighted and consume low resource while executing.

**Keywords:** Image processing, Deep Learning, Computer Vision, YOLOv4-Tiny, FastMOT, Wrong-Way Driving, Lane Detection, Hough Transform



## ACKNOWLEDGEMENTS

I would like to thank Dr. Chalernpol Charnsripinyo for supervising my research. He gave me the opportunity to join his group on the theme of a smart living platform. I would also like to thank my advisor Associate Professor Sasiporn Usanavasin for advising this research. They always encouraged me to think and grow as a researcher. Both gave me plenty of guidance and idea to design and develop this research work. Their support has been invaluable to this success. I also would like to thank other committee members, Dr. Seksan Laitrakun for his helpful comments and suggestions to improve the quality of this thesis.

This research is financially supported by Thailand Advanced Institute of Science and Technology (TAIST), National Science and Technology Development Agency (NSTDA), Tokyo Institute of Technology, Sirindhorn International Institute of Technology (SIIT), Thammasat University (TU) under the TAIST Tokyo Tech Program.

Pintusorn Suttioponpisarn

## TABLE OF CONTENTS

	Page
ABSTRACT	(1)
ACKNOWLEDGEMENTS	(3)
LIST OF TABLES	(7)
LIST OF FIGURES	(8)
LIST OF SYMBOLS/ABBREVIATIONS	(12)
CHAPTER 1 INTRODUCTION	
1.1 Background	1
1.2 Statement of Problem	3
1.3 Objectives	3
1.4 Thesis Outline	4
CHAPTER 2 REVIEW OF LITERATURE	
2.1 Wrong-Way Driving Detection	6
2.2 Object Detection	8
2.3 Object Tracking	9
2.4 Road Lane Detection	10
2.5 Embedded System	11
CHAPTER 3 WRONGWAY-LVDC FRAMEWORK	
3.1 Framework Flow	13
3.2 Framework Version	14
3.3 Input	15
3.3.1 Input Image	16
3.3.2 Vehicle Information	17

**CHAPTER 4 PROPOSED METHOD: ROAD LANE DETECTION**

4.1 Orientation Decision	19
4.2 Image Preprocessing	23
4.3 Area of Interest	26
4.4 Lane Line Decision	28

**CHAPTER 5 PROPOSED METHOD: WRONG-WAY DRIVING DETECTION**

5.1 Majority Based Correct Direction Detection	35
5.1.1 Correct Direction Validation Part	35
5.1.2 Detection Part	40
5.2 Distance-Based Direction Detection	42
5.2.1 Divide Road Area	43
5.2.2 Determine Area Type	44

**CHAPTER 6 INSIDE BOUNDARY IMAGE CAPTURING FEATURE**

6.1 Vertical Orientation Image Capturing	48
6.2 Horizontal Orientation Image Capturing	51

**CHAPTER 7 EXPERIMENTAL RESULTS**

7.1 Dataset	53
7.2 Vehicle Detection	53
7.3 Framework's Result	54
7.3.1 Road Lane Detection based on CCTV Algorithm	54
7.3.2 Distance-Based Direction Detection Algorithm	55
7.3.3 Inside Boundary Image Capturing Feature	58

**CHAPTER 8 CONCLUSION**

8.1 Limitations	60
8.2 Challenging Constraints	62
8.3 Summary of Thesis	65

<b>REFERENCES</b>	<b>67</b>
-------------------	-----------

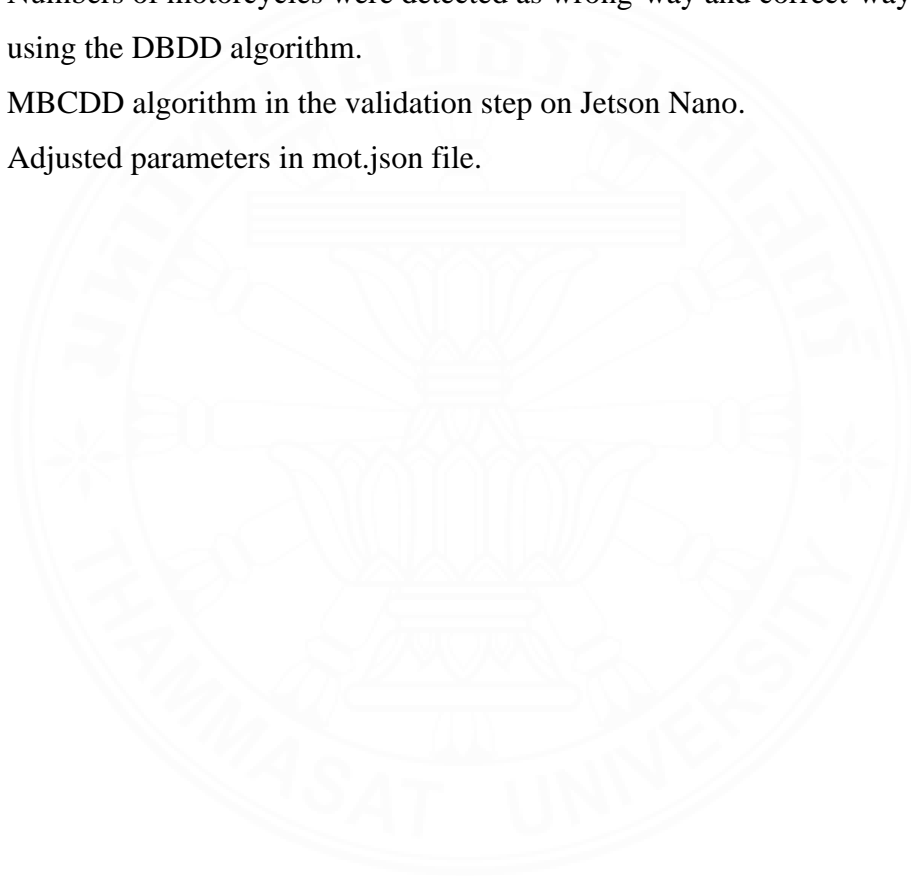
BIOGRAPHY





## LIST OF TABLES

Tables	Page
2.1 Comparison of wrong-way driving detection papers.	6
5. 1 Information about each unique vehicle ID.	37
7.1 Accuracy of the YOLOv4 Tiny model.	54
7.2 The number of start and stop points on each divided road area.	55
7.3 Numbers of motorcycles were detected as wrong-way and correct-way driving using the DBDD algorithm.	57
8.1 MBCDD algorithm in the validation step on Jetson Nano.	61
8.2 Adjusted parameters in mot.json file.	64



## LIST OF FIGURES

Figures	Page
1.1 Image evidence of many motorcycles driving in the wrong direction on Paholyothin road, in front of Bangkok University.	2
3.1 WrongWay-LVDC's framework flow.	13
3.2 System flow for different framework versions.	14
3.3 Roads that do not pass the condition (a) The construction is one the side road (b) Many vehicles are blocking the side road.	16
3.4 System flow on the method to obtain the vehicle information and image.	16
3.5 Vehicle Information plotting.	17
4.1 Camera view of CCTV (a) vertical orientation (b) vertical orientation (c) horizontal orientation.	19
4.2 System flow of RLB-CCTV algorithm.	19
4.3 The vehicle information was obtained after detecting and tracking all vehicles in the video for one minute (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two roads.	20
4.4 System flow of Orientation decision step.	20
4.5 Step to obtain the vectors to find the angle (a) Get vehicle information as an input (b) Set the lower point to be P1 and the upper point to be P2 (c) Draw a vector to the right where the initial point is P1 (d) Obtain the vehicle movement vector from P1 and P2 (e) Find the angle in degree.	21
4.6 All possible vectors and angles.	21
4.7 Angle value histogram plot (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	23
4.8 System flow of image preprocessing step.	23
4.9 Adjust the brightness on the input image (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	24
4.10 Apply CLAHE and Gaussian blur (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	25
4.11 Apply Histogram Equalization (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	25

4.12 Apply Canny edge detection (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	25
4.13 System flow of Area of Interest step.	26
4.14 Apply a convex hull to wrap around all start and stop points from vehicle information. (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Expand the edge of the convex hull shape (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.	27
4.15 Apply a bitwise mask between the area of interest and the final output from the image preprocessing step (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Remove the upper and lower boundaries (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.	28
4.16 System flow of lane line detection step.	28
4.17 Apply Hough transform to draw straight lines on the image. The red lines are the unwanted Hough lines we received. The green lines are the correct Hough lines we expected (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Remove the unwanted red Hough lines (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.	30
4.18 Form polygons (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Overlay start and stop point from vehicle information on the polygons (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.	33
4.19 Final output (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	34
5.1 System flow for validation part in MBCDD.	36
5.2 Detecting and tracking all vehicles for direction validation.	36
5.3 The direction of most vehicles for each lane of the road.	37
5.4 The method to find the angle value from the vehicle moving direction.	38
5.5 All possible angles for the movement direction.	38
5.6 Degree distribution for each lane of the road.	39

5.7 Screenshot of the system detecting vehicles driving direction (a) Correct direction in the area one detected (b) Correct direction in the area two detected (c) Wrong direction in the area one detected.	41
5.8 System flow of DBDD algorithm.	43
5.9 Overlay all points from vehicle information on the areas (a) Horizontal orientation; (b) Vertical orientation (c) Vertical orientation with two lanes. Overlay start points from vehicle information on the areas (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes. Overlay stop points from vehicle information on the areas (g) Horizontal orientation (h) Vertical orientation (i) Vertical orientation with two lanes.	44
5.10 Label the area and get the stopping reference point (a) Horizontal orientation with a stop area on the right (b) Vertical orientation with a stop area on the bottom (c) Vertical orientation with two lanes with opposite driving directions.	45
5.11 Length of detected road boundary from the RLB-CCTV algorithm (a) Vertical orientation (b) Horizontal orientation.	46
5.12 The road boundaries are cropped for a range of 3/10 to 7/10 (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.	47
6.1 System flow of IBI capturing feature.	48
6.2 Image coordinate system.	49
6.3 The screenshots when the CY bounding box is lower than YL.	50
6.4 The screenshots when the CY bounding box is above YL by 20 pixels.	51
6.5 The screenshots when the wrong-way driving vehicle is driving upward.	51
6.6 The screenshots when capturing vehicles driving in the wrong direction in the vertical oriented video.	52
7.1 Labeling the dataset with three classes: Motorcycle, car, and large vehicle.	53
7.2 Output images with road boundaries using the RLB-CCTV algorithm.	55
7.3 Screenshot of the DBDD algorithm while detecting vehicles driving in the correct and wrong direction.	58
7.4 Output images of the wrong-way driving vehicles using the IBI capturing feature (a) Vertical orientation road (b) Horizontal orientation road.	58

8.1 Detected vehicle movements (a) 1,000 frame length (b) 2,000 frame length (c)  
3,000 frame length.



## LIST OF SYMBOLS/ABBREVIATIONS

<b>Symbols/Abbreviations</b>	<b>Terms</b>
WrongWay-LVDC	Wrong-Way Driving Detection framework with features of Lane detecting, Validating the direction, Detecting wrong-way driving vehicles, and Capturing image evidence
RLB-CCTV algorithm	Road Lane Boundary Detection based on CCTV camera angle algorithm
MBCDD algorithm	Majority-Based Correct Direction Detection algorithm
DBDD algorithm	Distance-Based Direction Detection algorithm
IBI capturing feature	Inside Boundary Image Capturing feature

# CHAPTER 1

## INTRODUCTION

### 1.1 Background

Looking back a few years ago; according to the 2018 Global Status Report on Road Safety, Thailand has the highest motorcycle road traffic deaths in the world (Thongphat, 2019). The total compensation for all accidents is around 500 billion baht. About half of the motorcycle crashes are caused by: speeding, cutting in front of other vehicles, poor driving skills, and driving on the wrong side of the road. It is always better to prevent such bad driving behaviors from occurring, for example, authorizing stricter motorcycle driving license exams, building special lanes for motorcycles in urban areas, putting up warning signs for dangerous spots that frequently have accidents, or using modern technology to monitor, evaluate, and punish the drivers when they do not follow the law. As a result, the prevention will provide safer roads for all. For the best possible outcome, no death occurs from motorcycle accidents and reductions in compensation paid to the victims.

Motorcycles driving in the wrong direction is one of the main issues in Thailand traffic that causes accidents, especially in the urban area. There are three main reasons why this wrong-way driving behavior keeps occurring. First, some areas have a distant U-turn point which is not convenient for the drivers, especially for small vehicle drivers like motorcycles. Assuming vehicle drivers want to go back a few blocks on a one-way road. To make it legal, they have to drive forward many kilometers to make a U-turn and drive back to make another U-turn to reach the destination which is just a hundred meters away from the initial point. Therefore, it is much more convenient to break the traffic law by driving in the wrong direction, which saves more time and energy. Second, Thailand has weak laws and punishments, which makes people not afraid to break the law in general. Third, traffic police cannot cover every area to take surveillance and catch misbehaved drivers who break the traffic law. There are some areas where a lot of wrong direction drivers appear, but there are no traffic police who are in charge and on duty for surveillance due to a large number of vehicles driving per day.

For law enforcement, the traffic police have to observe driving behaviors on roads and give tickets to those who break the law. From May to June 2017, police officers took serious action to catch those who drove in the wrong directions throughout Bangkok (Bangkok Biz News, accessed 2022). The total number of wrong ways driving vehicles that were caught by the police was more than 4,000. However, arresting this high violation rate is a tedious task and costs high prices in terms of time and energy to stay on duty.

As a researcher, we see this gap where we could implement today's technology to help the officers identify those who break the traffic law automatically. Furthermore, the system we create could report information to the Traffic Police Division and Department of Land Transport, as the evidence is collected through the camera while the system is on surveillance.

The police officer, or any department who is concerned about road accidents, can use our system as a road surveillance monitor. The system can receive videos from any closed-circuit television (CCTV) that capture the road's view and process to capture those vehicles that drive in the wrong direction. For further implementation, in case the camera has a high resolution enough, it can detect the license plate of vehicles that break the law and escalate to further law enforcement and ticket fines. For example, some motorcycles are moving in the wrong direction, as shown in Figure 1.1. We are motivated to create a system to detect and alert when this kind of driving behavior is found.



**Figure 1.1** Image evidence of many motorcycles driving in the wrong direction on Paholyothin road, in front of Bangkok University.



## 1.2 Statement of Problem

There are a high number of accidents due to wrong-way driving vehicles, especially motorcycles, in Thailand. With the emergence of IoT, many organizations and region governors install CCTV cameras in their responsible areas for surveillance and evidence recording. CCTV cameras are substantially installed for surveillance and road traffic monitoring. The CCTV cameras can also record the driving behavior on road. Most CCTV cameras that record road traffic is likely to be installed at a high position such as on the skywalk or the electric poles to allow the camera to record a wide angle of traffic roads. We used this benefit of a CCTV camera wide-angle to observe and analyze the rich details that are hidden inside the videos. With the function of CCTV, it would record a lot of evidence of law-breaking driving behavior on road. If we could filter out only the moments that there are vehicles that drive in the wrong direction, it would be great information for the police to evaluate the dangerous level of that area. If the area has many wrong-way drivers, the police can authorize stricter surveillance, especially in that area. Therefore, we would like to create a system that can automatically detect wrong-way driving behavior from CCTV videos in real-time. The system can validate the correct direction for each road lane by itself. Our system aims for this kind of vehicle on the road as they mostly appear on the video: motorcycles, cars, and big vehicles (buses and trucks).

## 1.3 Objective

To create a system that can detect the wrong-way driving behaviors according to our motivation, we have set objectives and targets, which later on it is extended to other additional features to allow the system to be convenient to use. There are five objectives in this research.

1. To detect and track vehicles on the road to identify if the driving direction is correct on that specific road direction by using the videos from the CCTV camera's viewpoint
2. To propose an efficient correct driving direction validation algorithm on the road. The algorithm should identify the moving direction of each vehicle on the road and verify the correct and wrong direction automatically.

3. To create an algorithm that can identify the boundary of the road area from the CCTV viewpoint's perspective as an area of interest.
4. To capture images of the wrong-way driving vehicles as we can detect from the proposed system.
5. To implement the whole system onto an embedded system and be able to run the detection at real-time speed.

As we aim to detect the road lane boundaries, validate the correct moving direction of vehicles inside each road boundary, and detect the wrong-way driving vehicle. Next, we have considered the case where the users want to gather evidence of the vehicle that drives in the wrong direction in the form of image capturing from the input video. With this requirement, we have to make sure that the captured image must be clear and that the vehicles must arrive as close as possible to the CCTV camera position.

With all of these objectives, it represents each feature that the system could perform. Each objective is designed to link and execute after one another. Therefore, we would like to combine all objectives into a framework called "WrongWay-LVDC". LVDC stands for the main features that the framework can do, which are Lane detecting, Validating the correct direction, Detecting wrong-way driving, and Capturing the evidence images.

#### **1.4 Thesis Outline**

This thesis consists of seven chapters. Starts with Chapter one where we introduce our research with the background, statement of problem, objective, and the thesis's outline. In Chapter Two, we reviewed other research and papers that have related knowledge to our research's field. In Chapter Three, we explained our proposed framework along with the required input for the system.

After we have identified the features that will consist in our research, next, we proposed a solution for each WrongWay-LVDC framework's features. Beginning with the road lane detection method, where we proposed Road Lane Boundary Detection based on the CCTV camera angle (RLB-CCTV) algorithm. RLB-CCTV algorithm is a proposed algorithm where we used image preprocessing and the Hough transform method to solve the task in this part. More information about the RLB-CCTV algorithm

will be explained in detail in Chapter Four of this thesis. The next feature of the framework is to detect wrong-way driving vehicles inside the road lane, which is the main contribution of our research. In this feature, we proposed two algorithms that use different techniques. Each technique has its advantage and disadvantage. The first algorithm for wrong-way driving is the “Majority-Based Correct Direction Detection (MBCDD) algorithm” and another one is called the “Distance-Based Direction Detection (DBDD) algorithm”. As we tested these two algorithms, we concluded that the DBDD algorithm is more efficient in terms of validating and decided to use the DBDD algorithm as the main algorithm for wrong-way detection in our framework. More information about these two wrong-way validations will be explained in Chapter Five. The last feature is to capture the evidence of the wrong-way driving vehicles. In Chapter Six, we will explain more about the method that we proposed and used for capturing the moving vehicles in different location settings in each video, where those images captured must show the clearest shot of the vehicles. In this feature, we used the “Inside Boundary Image Capturing feature (IBI Capturing)”. After we have proposed different methods and algorithms for each feature, in Chapter Seven, we will show the result as we applied our algorithms to the real video data from CCTV. Lastly, in Chapter Eight, we will conclude our proposed framework’s result.

## CHAPTER 2

### REVIEW OF LITERATURE

#### 2.1 Wrong-Way Driving Detection

To detect vehicles with wrong driving directions, we need to understand and study the previous works that are related to this area. As we searched for other previously published papers, there are some researches with similar ideas and purposes that aim to detect vehicles in the wrong moving direction. Nevertheless, they are all presenting different interesting and unique techniques to create the system. After reviewing each paper, we found out that each paper has its advantages and disadvantages as shown in Table 2.1. (Monteiro, Ribeiro, Marcos & Batista, 2007; Usmankhujiev, Baydadaev & Kwon, 2020; Rahman, Ami & Ullah, 2020).

**Table 2.1** Comparison of wrong-way driving detection papers.

Papers	Monteiro, Ribeiro, Marcos, and Batista (2007)	Usmankhujiev, Baydadaev, and Kwon (2020)	Rahman, Ami, and Ullah (2020)
Step 1	Learn the orientation pattern of vehicles motion flow	Vehicle detection using YOLOv3	Vehicle detection using the YOLO algorithm
Step 2	Detect opposite moving object	Tracking using Kalman filter	Centroid tracking algorithm
Step 3	Validate from the appearance-based approach	Validation Entry-Exit algorithm	wrong way driving vehicle detection
Advantage	Focus only on the moving objects	Precisely identify the correct moving direction	The efficient proposed tracking method
Limitation	Occluded scenario	Manually specify the correct direction	Only one-way detection

In the first paper, Monteiro, Ribeiro, Marcos, and Batista (2007) have proposed a solution to detect the wrong driving direction of vehicles on highways using optical

flow. They separated the process into three stages to learn, detect, and validate the correct direction. For the first stage, the orientation pattern of vehicle motion flow is learned and used for detecting the opposite moving object direction in the second stage. Lastly, an appearance-based approach is being used to detect whether the moving objects are vehicles. The system has been tested on real highway traffic surveillance cameras under different weather conditions. The system can detect driving in the wrong direction successfully. Since the authors focused on detecting wrong-way drivers on the highway road where vehicles are not crowded and there are no distracting moving objects, the occlusion situations and roadside noise had not been mentioned and included. Moreover, the proposed system was not designed to track the vehicle individually. In this case, it is a challenge to identify each vehicle.

In the second paper, Usmankhujiev, Baydadaev, and Kwon (2020). developed a real-time system to detect vehicles traveling in the wrong way on the road from CCTVs. Three stages were proposed: Detection using You Only Look Once Version three (YOLOv3), Tracking using Kalman filter, and validation with their proposed algorithm “Entry-Exit”. They showed many comparisons of models and algorithms for both detection and tracking aspects and they selected the best algorithm for their task. For the Entry-Exit validation algorithm, the author had to identify manually which areas were considered incorrect or correct regions for entering or exiting. The result was 91.98% accuracy for detecting wrong-way driving vehicles.

In the third paper, Rahman, Ami, and Ullah (2020) proposed an automatic wrong-way vehicle detection system from CCTV footage. Their system is also divided into three stages: vehicle detection using the YOLO algorithm, centroid tracking algorithm, and detecting the wrong-way driving vehicles. As the authors showed the detecting result, the perspective of the camera on the road is on the road level and that caused a lot of vehicle occlusion. However, they achieved great accuracy in their results. The authors also mentioned that their system can only detect one side of the road which is their research limitation.

With the previous research we reviewed, we aim to create a system that can resolve those limitations other papers have found. Therefore, we have set our proposed wrong-way driving detection system’s objective as follows:

- The system can validate the correct direction by itself

- The system can detect wrong-way driving vehicles on multiple lanes
- At the end of detection, the system can report the exact number of wrong and correct driving vehicles from each lane
- The system should run at real-time speed

## 2.2 Object Detection

To detect many vehicles, some of which are occluded, in each frame of the video precisely and rapidly, we need to select the model that can satisfy our requirements. Mandal and Adu-Gyamfi (2020) have done comparative studies among different combinations of deep learning object detectors with the Deep SORT tracking algorithm. As a result, their study proved that for counting all vehicles on the roadway, YOLOv4 and Deep SORT, Detectron2 and Deep SORT, and CenterNet and Deep SORT were the most ideal combinations under various weather conditions and they can be further fine-tuned for other conditions of detection and tracking. The method they used for proving is by having each of the model combinations tested on a total of 546 video clips of one minute each. Some models' prediction returns a very high false positive and false negative. The combination of CenterNet and Deep SORT achieved an average accuracy of 95.96%, followed by 92.531% for Detectron2, and 90.98% for YOLOv4 and Deep SORT. With similar accuracies for detecting objects, speed of detection is also an important criterion to determine the most suitable model for this task. It is important to consider a fast model since the system we create aims to implement on an embedded system. Therefore, we are interested in YOLOv4 as it is proved by Bochkovski, Wang, and Liao (2020) to be one of the fastest object detector models with a real-time speed of ~65 FPS on Tesla V100 and state-of-the-art 43.5% AP (65.7% AP50) for the MS COCO dataset. Even if there is proof that YOLOv4 is accurate for detecting but to run object detection and tracking on embedded systems with small computation ability, YOLOv4 might not be the best model since it is big and consumes a large number of resources while detecting. Therefore, we decided to use YOLOv4-tiny instead. As Oltaen, Florea, Orghidan, and Oltean (2019) mentioned in their research, they adopted YOLOv3-tiny for real-time vehicle counting systems. YOLO-tiny has a faster training time and detection time due to the adaptation of the architecture from YOLO to be more lightweight.

### 2.3 Object Tracking

After detecting the vehicles, the next step is tracking. There are many traditional tracking algorithms and the Kalman filter (Gunjal, Gunjal, Shinde, Vanam & Aher, 2018) is one of them. The Kalman filter concept is to predict and compare the next frame objects' location in a video and try to update the prediction of the next frame based on the error it received from each prediction. Due to its accuracy, the Kalman filter is being used for many real-time tracking applications. Deep SORT (Wojke, Bewley & Paulus, 2017) is the latter generation and is one of the deep learning-based tracking approaches where it has adopted the tracking the idea from Kalman Filter. Deep SORT is an extension to SORT (Simple Online Real-time Tracker) with the improvement for detecting the uncertainty of object state estimation, such as object surface. Deep SORT can mostly overcome many tracking challenges such as object occlusion, tracking ID switches, tracking speed, running in real-time, and handling multiple objects in a video. Azhar, Zaman, Tahir, and Hashim (2020) applied Deep SORT in their research for their people tracking system. They showed that the accuracy of tracking results also depends on the object detection method as well. They compared YOLOv3 and YOLOv3-tiny detection results with Deep SORT as the tracking algorithm. YOLOv3 makes the tracking system accurate, but with a tradeoff, the average frame per second (FPS) is very low. In the meanwhile, YOLOv3-tiny showed the opposite result, with high FPS but lower accuracy. However, considering Deep SORT alone, it is a great candidate for the tracking algorithm and we selected it to use in our system.

In this research, we aim to create a system that has a suitable capability to be implemented on a small embedded system in real-time. Therefore, speed is one of the most important criteria to consider. In our previous experiment, we used Deep SORT for vehicle tracking. As we tried implementing the system on Jetson nano, the speed of the overall system is around two frames per second, which is too slow for real-time detection. However, FastMOT (Yang, 2020) seems to be a better choice for the tracking algorithm. FastMOT is a custom multiple object tracker on YOLO, Deep SORT, and KLT algorithm, which has two-stage trackers: Object detection and feature extraction. These two features will run sequentially, which makes the algorithm can track objects accurately. FastMOT can track objects with high speed even on a Jetson embedded

system because the algorithm itself is improved to run the detector and feature extraction every N frame and use KLT to fill the gap efficiently. This algorithm has been evaluated and can process up to 42 frames per second on the MOT17 challenge for multiple objects tracking benchmark.

## 2.4 Road Lane Detection

Several works research how to detect the road lane in the Advanced Driver-Assistance Systems (ADAS). To categorize the method from these researches, there are two main approaches for road lane detection: Deep learning-based and image processing-based. For the deep learning-based technique, the road lane images must be gathered and trained in the deep learning model. Liu, Chen, Zhu, and, Tan (2021) proposed the CondLaneNet algorithm to detect the road lane. The algorithm can also overcome the various conditions of road lines, such as dense lines and forks lines with high accuracy of 78.14% in the F1 score. Another deep learning-based method is by Chen, Xu, Pan, Cao, and Li (2021), who improved the existing model to become more efficient. They trained the model with the CUlane dataset. As a result, their model achieved up to 92.20% on F1 for the normal scenario. The deep learning-based method for lane detection is great for real-time detection and can return a high accuracy result. However, this method may require a higher computing performance than the computer vision-based method. Since our computing edge device has a limited resource for computing ability and we aim for the system to be computed at a real-time speed. therefore, image processing-based is more suitable for us in the lane detection algorithm.

There are many previous successful and interesting lane detection algorithms based on the image processing approach. The first one is from Andrei, Boiangiu, Tarbă, and Voncilă (2022), where the authors aimed to detect if the road is turning to the left, right, or going straight according to the video from the front vehicle's camera in ADAS. They preprocess each video frame with the Canny edge detector, Hough transform, and region of interest identification. In the last step, they try to find the parallelogram lines for drawing the region of interest instead of a trapezoidal one. As a result, their proposed method surpassed Sharma, Kumar, Gupta, and Kumar's (2021) accuracy result by 3%. The second paper is from Rakotondrajao and Jangsamsi (2019), who used Inverse



Perspective Mapping (IPM) to convert the camera perspective to a bird's eye view to be able to detect the four corners of the road lane in ADAS. The third is Franco, Santos, Yoshino, Yoshioka, and Justo (2021), who create the lane detection for ADAS using an embedded system. The system consists of three steps: pre-processing (reducing the noise), processing (extracting useful information), and post-processing (interpreting the information using computer vision). Their system uses the statistical technique to help detect the lane, which is computationally less intensive than neural networks and suitable for running on an embedded system. The fourth is Ghazali, Xiao, and Ma, (2012) who proposed three steps to detect the road lanes. Beginning with the selection part, where they remove the unnecessary area of the input image. Next is the image preprocessing using the proposed H-Maxima method. Lastly, the system detects straight and curve lines using an improved Hough transform which provides a faster and more accurate result than the original Hough transform. The last research is from Farag and Saleh (2018), where the Lane real-time detection (LaneRTD) system is created with the method of computer vision to detect road lanes in ADAS. This paper has stated a clear step for image preprocessing. As a result, the speed of lane detection achieved up to 11 FPS for detecting the lane in challenging videos.

## **2.5 Embedded System**

To create a system that can process in real-time speed, we have to consider the processing method. Assuming that we would like to install the system onto the one hundred CCTV, if we let the computation take place on a few computers where all the videos came from all CCTV via LAN wire, the bottleneck issue would happen on the computation part. But if the number of the computer is equal to the number of CCTV, that would be ideal but it is not efficient in terms of cost and energy consumption. If we use a cloud computing method, where all the video must be uploaded on the internet and the processing unit is on the cloud computing, it would require a lot of bandwidth to upload many videos from many different CCTV sources. Pudasaini and Abhari (2020) also mentioned this in their research as they developed the moving objects detection and tracking algorithm using the deep learning method. They showed that using edge devices can decrease latency time and reduce network bandwidth in their research. Therefore, one of the most efficient tools to process the system in real-time

with a large amount of data is by processing via an edge device. Gu, Ge, Cao, Chen, Wei, Fu, and Hu (2021) proposed Siamese Convolutional Network for object tracking running on edge-cloud architecture, which allows the tracking algorithm to process in real-time speed and is suitable for IoT devices. Nowadays, many edge devices can handle heavy processing tasks, such as image processing and deep learning with little power consumption in real-time speed. In terms of image processing, the device can be installed with a camera attached to it and act like a CCTV. Examples of popular embedded systems are Google Coral ([www.coral.ai](http://www.coral.ai), accessed 2022), Raspberry Pi ([www.raspberrypi.org](http://www.raspberrypi.org), accessed 2022), and NVIDIA Jetson ([www.developer.nvidia.com](http://www.developer.nvidia.com), accessed 2022), where each brand has its advantages. Zualkernan, Dhou, Judas, Sajun, Gomez, and Hussain (2022) have compared several brands of edge devices. They show that Raspberry Pi consumes the least energy and Jetson provides the fastest result. As our research is focused on image processing and deep learning for the wrong-way driving detection, NVIDIA Jetson would be our top choice for embedded systems. There are many versions for Jetson. Each version has its unique feature and the level of computing ability and power consumption depends on the intensity of each project. In our research, we aim to detect the driving direction of the vehicles, which is straightforward and does not require high computation power. Therefore, we selected the NVIDIA Jetson Nano Developer Kit ([www.developer.nvidia.com/embedded/jetson-modules](http://www.developer.nvidia.com/embedded/jetson-modules), accessed 2022), the lightest version of Jetson that consumes the least energy with adequate power for computing. Especially if we would like to implement our system in many locations, the cost is one of the main factors we should consider too. In this research, we developed the framework based on the case that the computing unit is based on a PC as our main point in this research is to propose our idea of this framework. However, we also consider the requirements that allow the framework can be run on edge devices as well.

## CHAPTER 3

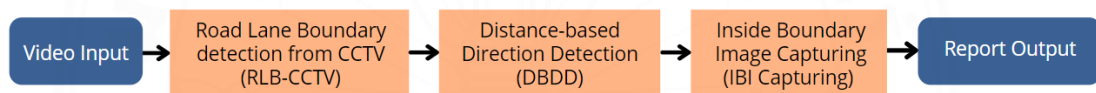
### WRONGWAY-LVDC FRAMEWORK

#### 3.1 Framework Flow

We have mentioned the proposed algorithms and features inside our WrongWay-LVDC framework, which are as follows:

- Road Lane Boundary Detection based on CCTV camera angle (RLB-CCTV)
- Distance-based Direction Detection Algorithm (DBDD)
- Inside Boundary Image Capturing Feature (IBI Capturing)

In our proposed framework, these algorithms and features will be called and executed after one another. Therefore, the framework flow for WrongWay-LVDC is shown in Figure 3.1 below.



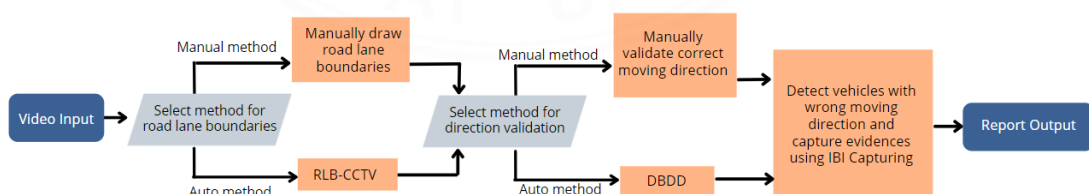
**Figure 3.1** WrongWay-LVDC's framework flow.

In the framework flow shown in Figure 2, to begin with, the system will detect the road boundaries using the RLB-CCTV algorithm. Next, the wrong-way driving validation and detection system will validate the correct moving direction of the vehicle on each detected road boundary using the DBDD algorithm. After the validation, the system will generate the threshold value that will be used to measure the correct driving direction in each lane and the detection part will be performed. As the system detect and track each vehicle in each lane in the detection part, it will compare the moving direction of the vehicle and whether it corresponds to the given threshold of each lane. If the vehicle driving correctly or wrongly, the accumulating counters will count the number of correct and wrong driving vehicles. After the detection part ends, the counters will be summarized and shown to the user as the report let the user evaluate the traffic of wrong-way driving level. If the intensity of wrong-way driving is high, compared to the correct driving, that area will be marked as a dangerous area and send the alert along with the vehicle counter-report as an output of the system to the users.

The input of the framework is expected to be recorded in video or streamed video from CCTV cameras. However, we have tested our framework with the recorded video using the smartphone. In the ideal situation of detecting streaming videos, we intended to have the input videos of 60-minute length for each video, which means, for each location, the detection result will be summarized and gathered every hour. Considering the gap between each streamed video, after a video reached up to 60 minutes, the system will be rebooted. During the reboot time where it might take around 10 seconds running on a computer and around 20 seconds running on Jetson Nano, the wrong-way driver detection framework might miss some of the detection that may occur. However, comparing this missed number of detections to the whole detection of 60 minutes, the missing value is insignificant. The framework can still display and capture the main result of the detection.

One of the reasons for dividing the video into 60-minutes time slots is due to the data management purpose. As the system detects the vehicles driving at a real-time speed, either driving in the correct or wrong direction, the unique ID will be given to each vehicle and will be labeled on top of the vehicle bounding box. Assuming the case where the system has been run for several hours, the unique ID will keep increasing endlessly. To restart the system every hour, the unique ID will also be restarted and the detecting data will be summarized. This is to allow the system to work faster with a lesser load in the memory.

### 3.2 Framework Version



**Figure 3.2** System flow for different framework versions.

The main purpose of our framework is to let each algorithm and feature executed automatically after one another. However, there are some concerns about the performance of our system when it meets some unusual scenarios that might be the limitation of our algorithm. In such a case, our system might not be accurate such as

the RLB-CCTV algorithm's limitation, where it cannot detect the road under certain conditions. As a final result, we still want our system to work in its best performance and function properly even if there are some unwanted conditions happened. To make it possible, we design other versions of our framework where the user can manually draw the road lane boundary instead of using the RLB-CCTV algorithm and validate the correct direction instead of using the DBDD algorithm. The framework versions have a workflow as shown in Figure 3.2.

### 3.3 Input

The concept of our framework is to receive video inputs from CCTV. Normally, the CCTV will record video for 24 hours a day and record the video periodically with a length of about one hour per video clip. We tried to obtain the videos that imitate the CCTV's angle to test with our proposed system. As we observe, most CCTVs are likely to be installed somewhere high above the roads, such as on the skywalk or the electric pole. Therefore, we visited many skywalks around the city and recorded the sample videos with a length of around five minutes.

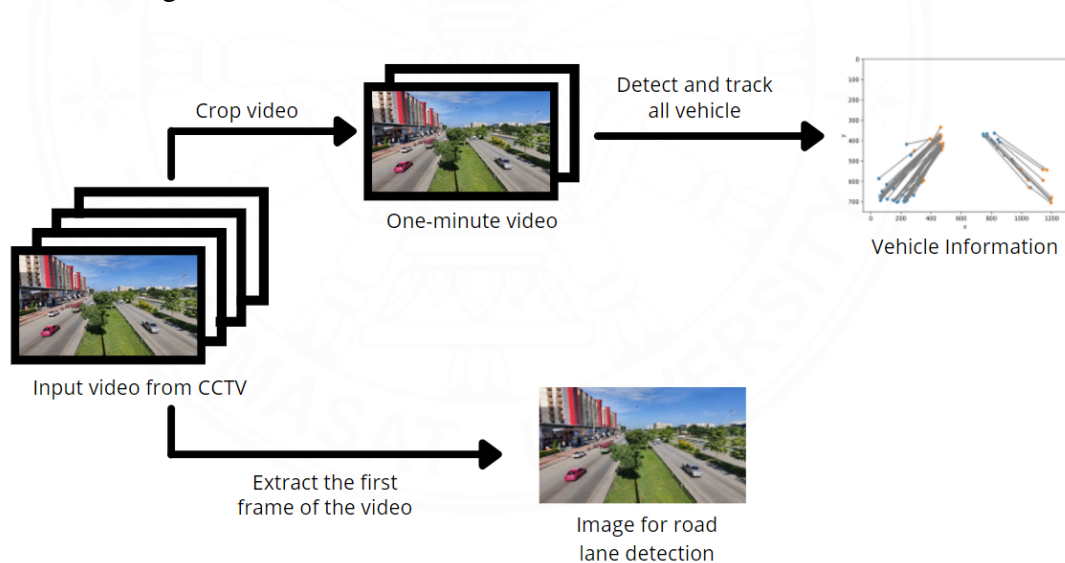
After we gather some video data of the road traffic in Thailand, we can observe several challenging factors to create our road lane detection algorithm. For example, some road is under construction, the road line color is pale, and the angle of the CCTVs are different. With the various condition of the road lane that we have observed, the road with the unusual conditions might be our system's limitation where our system might not be able to detect the lane line on those roads properly. Figure 3.3 shows the images of road lanes that do not have a clear lane line. Therefore, we need to set a clear scope to select the normal road condition to be used with our proposed system. We mainly aim to apply this system to the main traffic roads with these given conditions:

- The road must be a straight line.
- The road must have a clear side road line or a clear footpath line.
- There should not be any objects blocking the side of the road.
- The road must not be under construction.
- The road should not have disturbing shadows from the side road objects.



**Figure 3.3** Roads that do not pass the condition (a) The construction is one the side road (b) Many vehicles are blocking the side road.

After we collect the video input, the videos will be further analyzed, processed, and transformed into many other types of input. To clearly explain the process, the input will be transformed into two other inputs, which are: An image input and vehicle information. In Figure 3.4, we show the system flow of how our system can generate more data using the CCTV video.



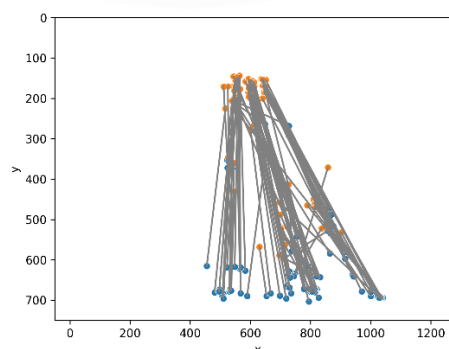
**Figure 3.4** System flow on the method to obtain the vehicle information and image.

### 3.3.1 Input Image

A still image input will be used in the road lane boundary detection algorithm for the image processing method to detect the road lane. This image input will be derived from extracting the first frame of the input video using the OpenCV library ([www.opencv.org](http://www.opencv.org), accessed 2022) in the Python programming language.

### 3.3.2 Vehicle Information

One of the most useful inputs we use in many systems of our proposed framework is vehicle information. It is the information that provides the location of each vehicle driving inside the video frame. There are 3 steps to obtain this input. First, crop the first one minute out of the main input video. Second, detect and track all vehicles moving inside the video frame regardless of the road lane. The object detection algorithm we use is YOLOv4-Tiny as developed from YOLOv4 (Bochkovskiy, Wang & Liao, 2020) to become smaller and suitable for running on small embedded systems. For the tracking algorithm, we chose the FastMOT algorithm (Yang, 2020). FastMOT is improved from the Deep SORT tracking algorithm, where it is revised to be more suitable to run on small devices without bottleneck issues. The technique of FastMOT is to run the detector and feature extraction every N frame and use KLT to fill the gap efficiently. In the third step, we will analyze the data we received. After all the vehicle has been tracked, the output will look like the image shown in Figure 3.3, where the plot is the summary of all movement vehicle within the time length of one minute. Each detected vehicle ID's information will be stored and displayed in a plot after the video is finished running. The orange dots inside the plot graph represent the location where each vehicle ID first appears inside the video frame or the start points. The blue dots represent the last location that the vehicle ID has been tracked before the vehicle left the video frame or the stop points. The black lines linked between the orange and the blue dot represent the movement direction each vehicle went. In Figure 3.5, we can roughly tell that there is one road lane appearing in this video frame, where most of the vehicle in the lane is driving downward.



**Figure 3.5** Vehicle Information plotting.

## **CHAPTER 4**

### **PROPOSED METHOD: ROAD LANE DETECTION**

As we plan our system to be able to detect the vehicle driving direction on multiple road lanes at the same time, therefore, we need to specify the boundary of each road lane for these given reasons:

1. Identify the area of interest to remove the side road noises, such as vehicles parking or bicycle riding outside the interested road lane
2. Reduce the confusion in the system as we track vehicles on multiple lanes.

The easiest and most accurate method to identify the road boundary of interest is the manual drawing from users. However, if we plan to install the system on many CCTV, we have to individually draw the lane boundaries of interest for each of the road lanes. In this case, automatic drawing would be a great help to identify the road boundaries on multiple CCTV. As a result, this inspired us to create another algorithm that can automatically identify the road lane boundaries inside the video frame using the image processing technique.

As we have mentioned earlier, there are several research papers on lane detection based on the image processing method. Most of the research has similar main steps to achieve the result: image preprocessing, select area of interest, and lane line decision. Moreover, there are many interesting techniques we can adopt and redesign to suit our research purpose. For example, the technique of Canny edge detection (Canny, 1986) in the image preprocessing step was used in the papers from Andrei, Boiangiu, Tarbă, and Voncilă, (2022), Franco, Santos, Yoshino, Yoshioka, and Justo (2021), and Farag and Saleh (2018). This technique produces an output where it focuses only on the contrast edge of the object inside the image, which makes the road lane clear, outstanding, and easy to detect. Another interesting technique is Hough transform (Duda & Hart, 1972). All of the research papers use Hough transform to make a decision and draw the line of the road lane. The last example technique is Gaussian Blur (Gedraite & Hadad, 2011) where Andrei, Boiangiu, Tarbă, and Voncilă, (2022), Franco, Santos, Yoshino, Yoshioka, and Justo (2021), and Farag and Saleh (2018) use in image preprocessing to reduce the excessive noise from the image.

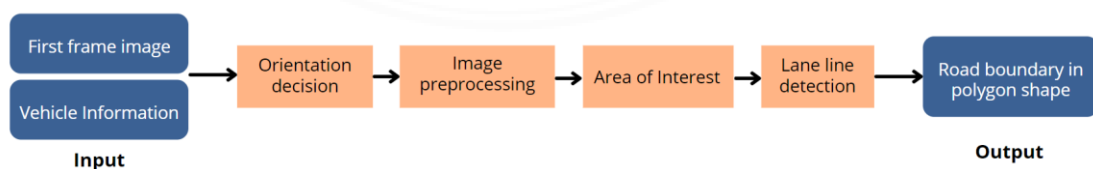


From the research we reviewed, all of them are designed for ADAS. Our research has a different purpose as we designed for the CCTV camera angle. Mostly, the setting of the CCTV camera has a high and center position to view the road at a clear angle. With this position, the CCTV will see the road in the position of driving up and down as shown in Figures 4.1 (a) and (b). However, we also consider the minor case where the CCTV is installed on the side of the road too, which the angle the CCTV will see will be in another position as shown in Figure 4.1 (c).



**Figure 4.1** Camera view of CCTV (a) vertical orientation  
(b) vertical orientation (c) horizontal orientation.

These three cases of the camera angle have an impact on us creating the lane detection algorithm. Our proposed RLB-CCTV algorithm consists of four main parts: orientation decision, image preprocessing, area of interest, and lane line decision. The input for this algorithm will consist of image input and vehicle information. The system flow of the overall algorithm is shown in Figure 4.2.

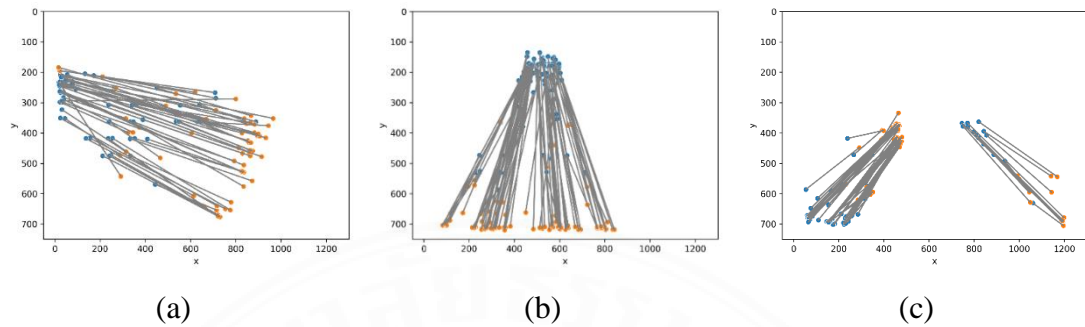


**Figure 4.2** System flow of RLB-CCTV algorithm.

#### 4.1 Orientation Decision

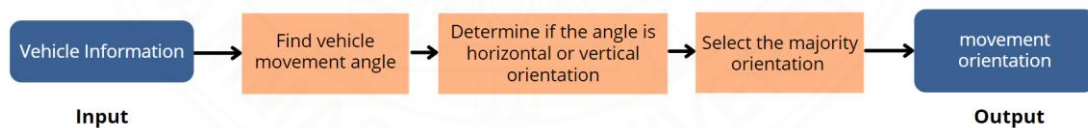
In the first step of the RLB-CCTV algorithm, the system will determine the lane orientation of the received video whether the lane is showing horizontal or vertical. The

input of this step is the vehicle information, which we show the example in Figures 4.3 (a), (b), and (c).



**Figure 4.3** The vehicle information was obtained after detecting and tracking all vehicles in the video for one minute (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two roads.

From the above figures, we can instinctively tell that Figure 4.3 (a) contains a road lane with a horizontal angle, Figure 4.3 (b) contains a vertical one, and Figure 4.3 (c) contains the vertical one with two road lanes. However, we have to create a method to let the system identify the orientation automatically by itself, where our proposed system flow is as shown in Figure 4.4 below.

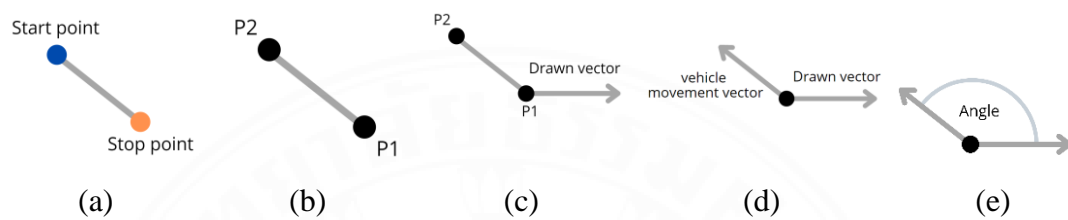


**Figure 4.4** System flow of Orientation decision step.

The main idea of our solution is to find the majority of each vehicle's movement angle and set the road orientation of that input video. To understand the concept of the vehicle's movement angle, we have to understand the basic concept of the angle between two vectors on the cartesian coordinate system. The angle has two types: radius and degree.

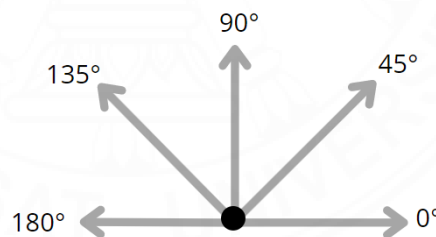
The two vectors we use in our calculation are obtained from the following sources: The vehicle movement and the imaginary horizontal drawn vector. To obtain the vehicle movement vector, the initial point of the vector must be either the start or

stop vehicle location where it has a lower x value on a plot graph as the pseudo-code explained in Algorithm 1 on lines number 3 to 9. The reason for this setting is to remove the case where the vector is pointing downward, which reduces the confusion when identifying the road orientation range. The example demonstration of how to obtain the two vectors and the angle is shown in Figure 4.5 below.



**Figure 4.5** Step to obtain the vectors to find the angle (a) Get vehicle information as an input (b) Set the lower point to be P1 and the upper point to be P2 (c) Draw a vector to the right where the initial point is P1 (d) Obtain the vehicle movement vector from P1 and P2 (e) Find the angle in degree.

In this case, we will use the degree system with the range of 0 degrees to 180 degrees as the direction of the vector indicate the angle and degree as shown in Figure 4.6.



**Figure 4.6** All possible vectors and angles.

After we obtain the angle for each vehicle movement in the vehicle information, next, we have to set the range of the angle where we will consider it as a horizontal orientation or vertical orientation. The condition we set is as shown in Algorithm 1 on lines number 11 and 13.

---

**Algorithm 1.** Pseudo Code for the Orientation Decision Step

---

**Input:** Vehicle Information

**Output:** Road Orientation

**Auxiliary Variables:** *Horizontal Count, Vertical Count*

---

---

**Initialization:** *Horizontal Count = 0, Vertical Count = 0*

**Begin Orientation Decision Algorithm**

```

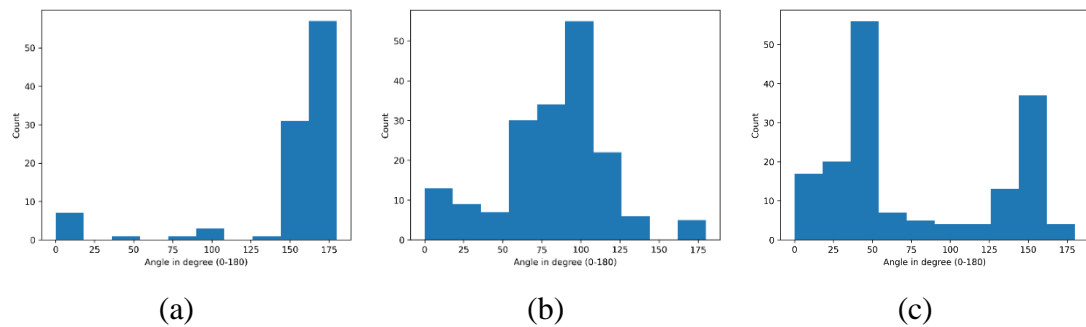
1  for (Each Vehicle Movement  $\in$  Vehicle Information) do
2      Start location, Stop location = extract information (Each Vehicle Movement)
3      if (Start location (y) < Stop location(y)) then
4          Point 1 = Start location
5          Point 2 = Stop location
6      else
7          Point 1 = Stop location
8          Point 2 = Start location
9      end if
10     Each Angle = Find Angle (Point 1, Point 2)
11     if ((Each Angle <= 180) AND (Each Angle > 155)) OR ((Each Angle >= 0)
12     AND (Each Angle < 35)) then
13         Horizontal Count = Horizontal Count + 1
14     elseif (Each Angle >= 35) AND (Each Angle <= 155) then
15         Vertical Count = Vertical Count + 1
16     end if
17 end for
18 if Horizontal Orient > Vertical Orient then
19     Road Orientation = Horizontal
20 else
21     Road Orientation = Vertical
22 end if
23 return Road Orientation

```

**End Orientation Decision Algorithm**

---

After finding the angle of all vehicle information, the result is shown in the histogram plot where Figure 4.7 (a) is the histogram result of Figure 4.3 (a), Figure 4.7 (b) is the result of Figure 4.3 (b), and Figure 4.7 (c) is the result of Figure 4.3 (c). The angle result corresponds to the method we have set, the range between 0 to 180. As well as the result of finding the road orientation, where Figure 4.7 (a) is a vertical orientation and Figure 4.7 (b) is a horizontal orientation. However, Figure 4.7 (c) is quite hard to identify by eyes on which the orientation is it, but as we compute it with our algorithm, it is a vertical orientation.

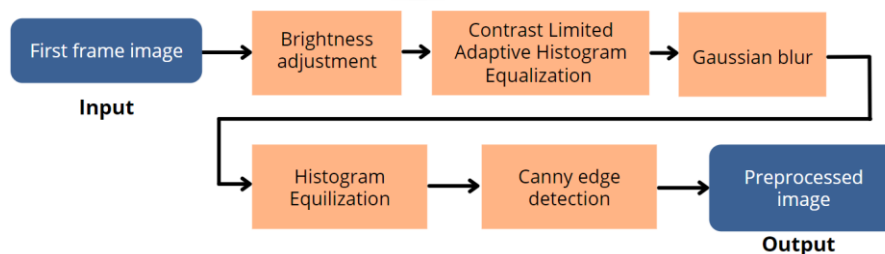


**Figure 4.7** Angle value histogram plot (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.

## 4.2 Image Preprocessing

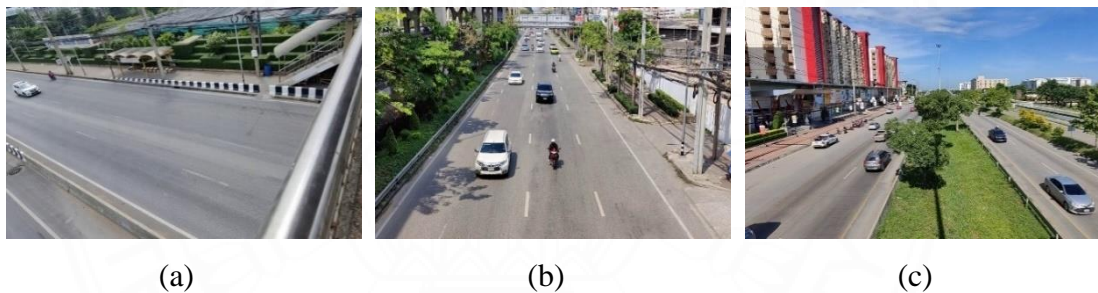
In this step, the system will perform the image preprocessing onto the input image to extract only the major details, especially the road lanes, and remove the noises that re-side inside the image. The input image will be preprocessed with a total of five techniques as shown in Figure 4.8. We will explain each technique in each step of the image preprocessing method.

The first step is brightness adjustment. The system will check the brightness level of the input image by converting the image into grayscale and get the average pixel brightness value using the ImageStat library module in Python (ImageStat Module, accessed 2022). If the input image originally has a low level of brightness, the output value will be near zero. If the image has a high level of brightness, the output value will be near 255. To increase the brightness of the image, it will show some hidden detail that was dimmed and dark. However, the brightness should be increased according to the original image condition.



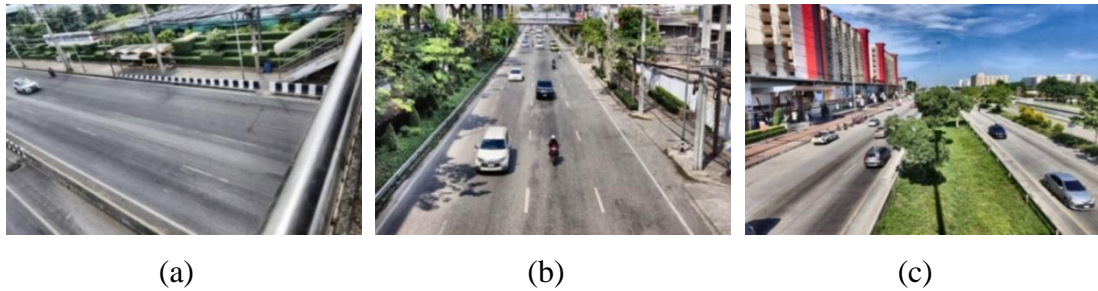
**Figure 4.8** System flow of image preprocessing step.

To increase the brightness of the image, it will show some hidden detail that was dimmed and dark. However, the brightness should be increased according to the original image condition. If the brightness in the image is already high, we should not increase the brightness on that image anymore because that will cause the over brightness exposure. Therefore, the brightness value will be subtracted by the threshold value that we have set at 130. We have tested with several images to make sure that 130 is the adequate level of brightness so that the image will not be too bright or too dim. After we find the difference between the brightness value and the threshold value, we will use that difference value to increase the brightness of the original image by converting the image into the HSV system (HSL and HSV, accessed 2022) and increasing the brightness according to the difference value we found. The result of brightness adjusting is shown in Figures 4.9 (a), (b), and (c)



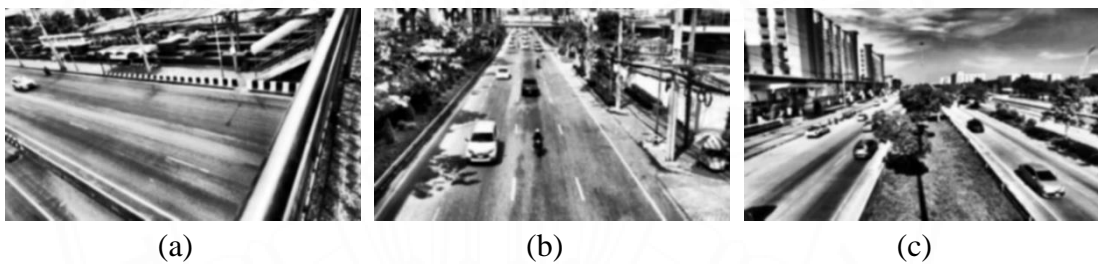
**Figure 4.9** Adjust the brightness on the input image (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.

The second step is to increase the global contrast to the image by using Contrast Limited Adaptive Histogram Equalization (CLAHE) (Yadav, Maheshwari & Agarwal, 2014) to make the road line look clearer. Followed by the third step, Gaussian Blur, where this technique will exclude some unnecessary small details not to be drawn in the edge decision step. With the OpenCV library, we have to apply Gaussian Blur 6 times to make the road image cover and hide all small detail. The result of steps two and three are shown in Figures 4.10 (a), (b), and (c).



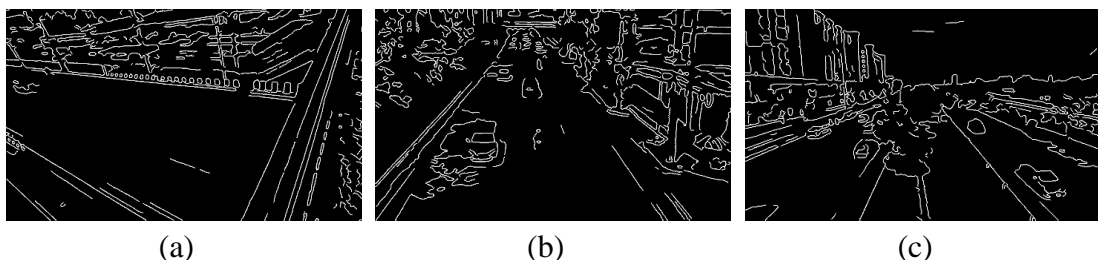
**Figure 4.10** Apply CLAHE and Gaussian blur (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.

The fourth step is to add Histogram Equalization (Patel, Maravi & Sharma, 2013) in OpenCV. The image needs to be converted into grayscale before applying this technique. As a result, as shown in Figures 4.11 (a), (b), and (c), the image will be increased its contrast to another level and the main detail is very clear in this stage.



**Figure 4.11** Apply Histogram Equalization (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.

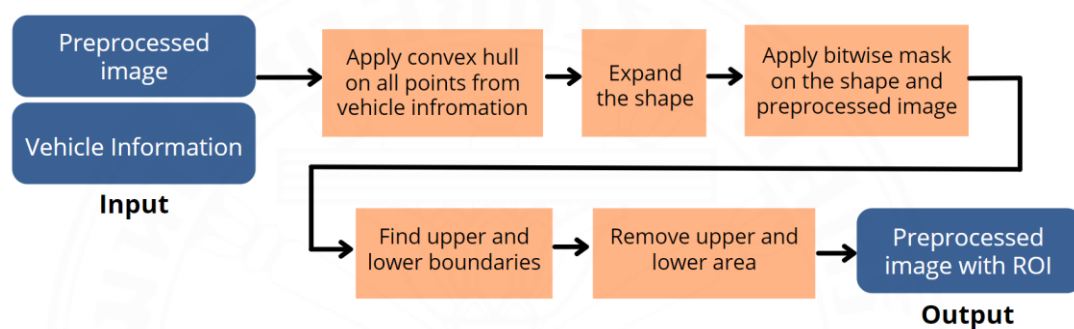
The fifth step is applying the Canny edge detection (Canny, 1986) to reduce the noise and only focus on the edge of the object. The final output from the image preprocessing step is shown in Figures 4.12 (a), (b), and (c).



**Figure 4.12** Apply Canny edge detection (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.

### 4.3 Area of Interest

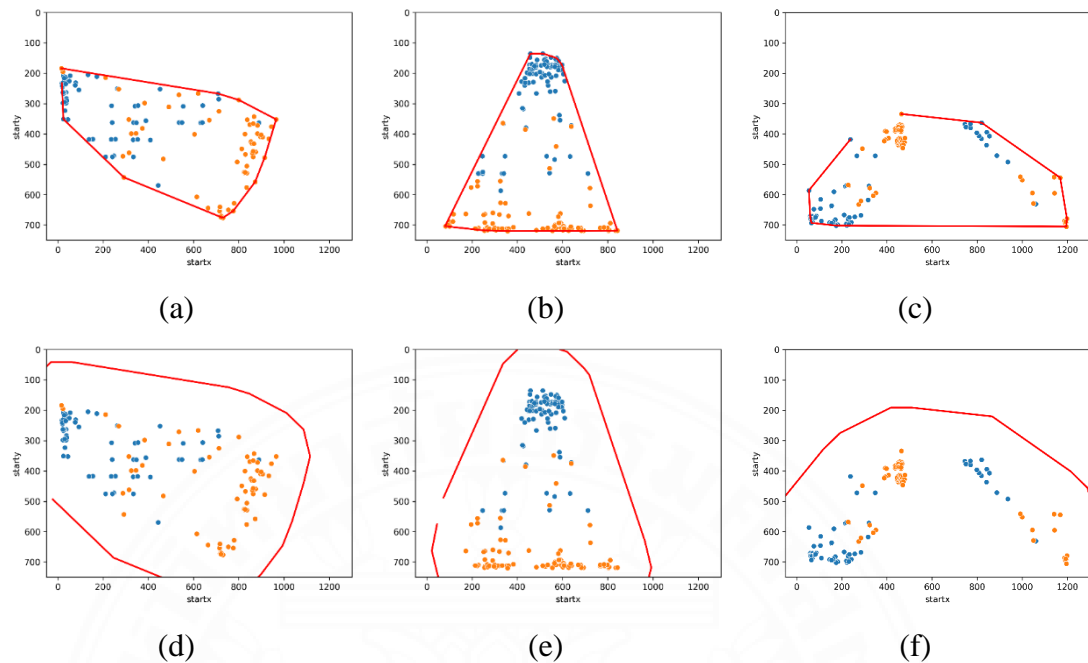
After we have the preprocessed image, we can see that the image still includes some side road areas where it will not contribute to our algorithm's purpose. These unwanted areas will decrease the lane line detection accuracy. The simplest method is to identify the Area of Interest (AOI). However, each road image has a unique position of AOI, so we should not fix the AOI for all images. The system flow for this step is shown in Figure 4.13.



**Figure 4.13** System flow of Area of Interest step.

The vehicle information allows the system to know the location where the vehicles are likely to drive. The vehicle drives on roads; therefore, the roads are our target AOI. As an assumption, we will focus on the area where vehicles are driving inside the video frame, and that area will be used as the AOI. To begin with, we use the technique of the convex hull (Sharif, 2011), which is the method to draw the lines that connect the dots that are positioned in the outer area of the geometric shape. We apply the convex hull onto the vehicle information we have, to draw a shape that wraps around all points (start point and stop point of the vehicles) as shown in Figures 4.14 (a), (b), and (c). The red line shape that is drawn will be considered as the area where the vehicles appear. However, those areas do not include the road sideline as the vehicle will not drive on the side road. Then, we need to increment the convex hull shape we drew to be larger by applying the Buffer points method and drawing a convex hull on the new buffer points. As a result, the areas are expanded and cover the road line as shown in Figures 4.14 (d), (e), and (f), where we will use this area as the AOI.



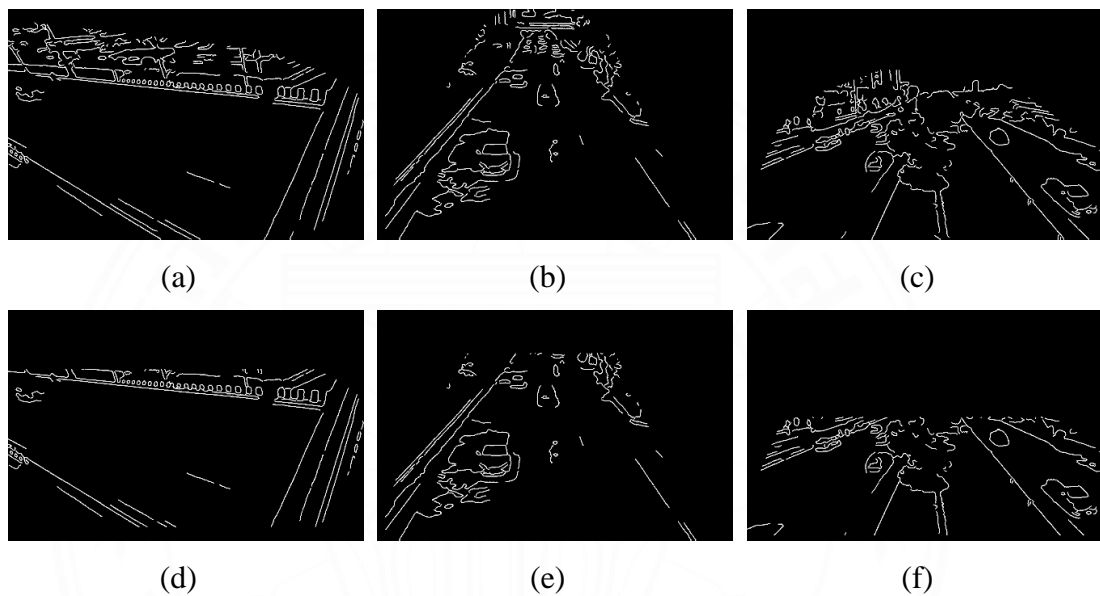


**Figure 4.14** Apply convex hull to wrap around all start and stop points from vehicle information. (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Expand the edge of the convex hull shape (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.

Since we have the AOI area, the next step is to overlay the AOI area onto the output image from the image preprocessing part by using the technique of OpenCV called bitwise mask (OpenCV Bitwise AND, OR, XOR, and NOT, accessed 2022). The bitwise mask will select to keep the part of the image we want and discard the rest. The result of this step is shown in Figures 4.15 (a), (b), and (c), where we can see that these two images have different locations and sizes of the AOI, and the drawn AOI fit to show the road area only for that image.

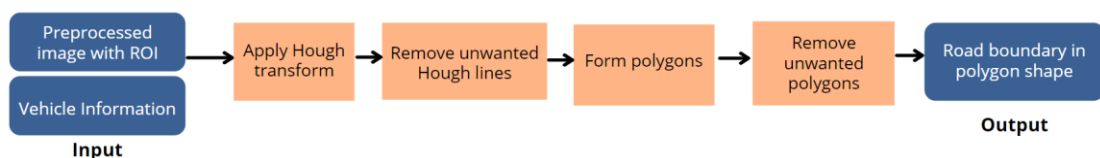
At this stage, we can see that in Figure 4.15 (a) remain some sideroad noises on the top, as well as in Figure 4.15 (b) where the top part of the image still has some noise. To remove those noises, we use the upper and lower boundary technique as once mentioned in Ghazali, Xiao, and Ma, (2012)'s the paper where the author uses this same technique to remove the sky area of the image. To find the upper and lower boundary, we use the vehicle information, where we get the points, either start or stop points, with

the highest and lowest y value in the cartesian coordinate system. Those points will be drawn a line over in a horizontal line and those lines are called the upper and lower boundary. Using the same technique, we, once again, apply the bit-wise mask of the upper and lower boundaries onto the image. The final result will be shown in Figures 4.15 (d), (e), and (f). The noises are removed and now the image is ready to apply to the next step of the RLB-CCTV algorithm, the lane line decision.



**Figure 4.15** Apply a bitwise mask between the area of interest and the final output from the image preprocessing step (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Remove the upper and lower boundaries (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.

#### 4.4 Lane Line Decision



**Figure 4.16** System flow of lane line detection step.

In lane line detection, the algorithm we use to detect the line is the Hough transform algorithm [28]. However, there are other decisions to make to select the most accurate Hough line, where the step is as shown in Figure 4.16. We set the system to draw 12 Hough lines in one image, to make sure all expected lane lines will be drawn.

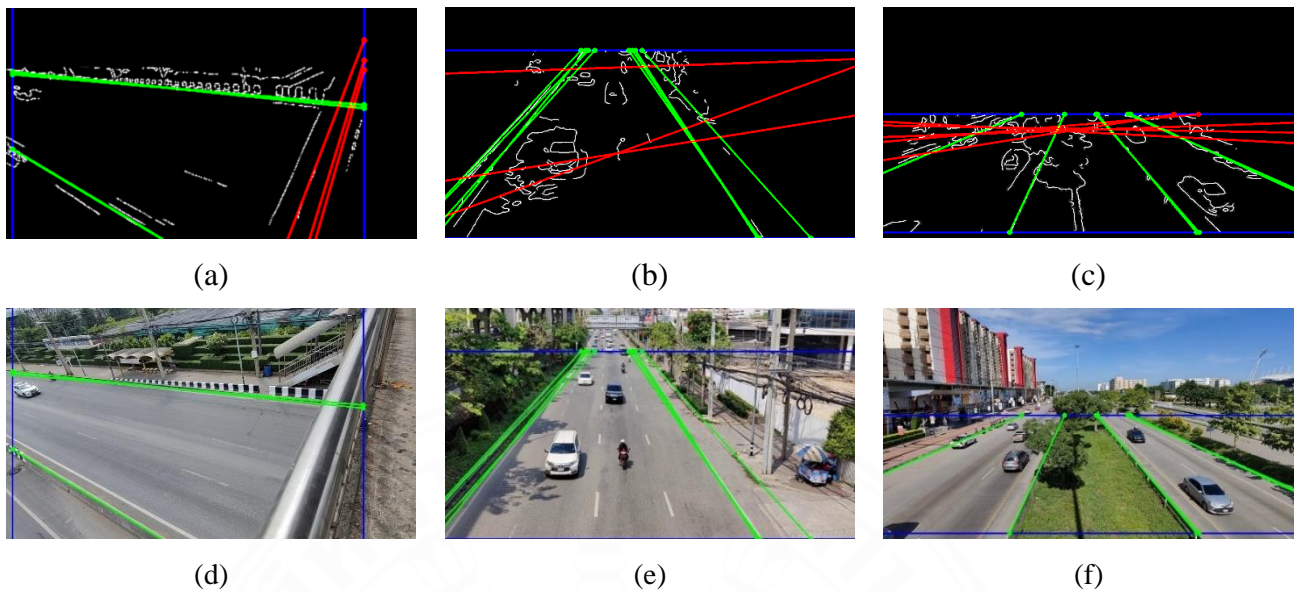
However, some unwanted Hough line has been drawn along the way, as shown in Figure 4.17 (a), (b), and (c). The red Hough line is the line where we detect it as an incorrect drawing. The method we filter those lines is based on the orientation of the lane, which we retrieved from the orientation decision. The system will check the slope value of each line, where the slope is calculated from the two endpoints of the Hough line. If the orientation is vertical like in Figure 4.17 (b) or (c), the blue color upper and lower boundaries will be drawn and the endpoints of the Hough lines will meet there. The slope of the accepted Hough line in the vertical orientation must meet this condition.

$$\begin{aligned} & |(y \text{ value of endpoint } 2 - y \text{ value of endpoint } 1) * 2| \\ & \text{is greater than } |x \text{ value of endpoint } 2 - x \text{ value of endpoint } 1| \end{aligned} \quad (4.1)$$

If the orientation is horizontal like in Figure 4.15 (a), the left and right boundaries will be drawn and the endpoints of the Hough lines will be extended to meet the two blue boundary lines. The slope of the accepted Hough line must meet this condition.

$$\begin{aligned} & |(x \text{ value of endpoint } 2 - x \text{ value of endpoint } 1) * 2| \\ & \text{is greater than } |y \text{ value of endpoint } 2 - y \text{ value of endpoint } 1| \end{aligned} \quad (4.2)$$

The red Hough line we draw in Figures 4.17 (a), (b), and (c) is an example to show the lines that do not meet the above condition we stated. As an output, the accepted Hough lines are drawn on the original input image, to compare and check the correctness as shown in Figures 4.17 (d), (e), and (f). However, we can see that several accepted lines are drawn on the image. We can remove the duplicated line but which lines should be removed. If we remove the inner lines and keep only the outer lines, that would work only in the case in Figures 4.17 (d) and (e), but would not work in the case where there is two-way traffic like in Figure 4.17 (f).



**Figure 4.17** Apply Hough transform to draw straight lines on the image. The red lines are the unwanted Hough lines we received. The green lines are the correct Hough lines we expected (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Remove the unwanted red Hough lines (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.

To design the method that can cover several cases, the next step is to draw the polygon area from the accepted Hough line. The technique that we proposed has the following method. First, we gather the locations of all intersection points between Hough lines and boundary lines. If the orientation is vertical, we will gather the x value of all intersection points, and if the orientation is horizontal, we will gather the y value of all intersection points. In the case of vertical, let's represent a list of the x values of the intersection point in the upper boundary with UpperX, where  $upperx_i$  is each x value contained inside UpperX:

$$UpperX = [upperx1, upperx2, upperx3, upperx4, \dots, upperxn] \quad (4.3)$$

Let's represent a list of the x values of the intersection point in the upper boundary with LowerX, where  $lowerx_i$  is each x value contained inside LowerX:

$$LowerX = [lowerx1, lowerx2, lowerx3, lowerx4, \dots, lowerxn] \quad (4.4)$$

Both UpperX and LowerX lists are sorted ascendingly. In the vertical orientation, the y value of the lower and upper boundaries will be fixed, as the boundary is a horizontal line. Therefore, the y value is a constant for all intersections of the lower and upper boundaries. In this case, we assign the y value of the upper boundary as UpperY and the y value of the lower boundary as LY. To create a four-corner polygon, we select the four points from the two lists, every two values from each list as the pseudo-code shown in Algorithm 2. From Algorithm 2 Line 7, it represents the method to create the polygon by using the Shapely geometry python library (The Shapely User Manual, accessed 2022). This method requires only four points as corners to draw a polygon.

---

**Algorithm 2.** Pseudo Code for Polygon Forming in Vertical Orientation

---

**Input:** UX, LX, UY, LY

**Output:** Polygon corners

Auxiliary Variables: Polygon\_list

Initialization: Polygon\_list = []

**Begin Polygon Forming in Vertical Orientation**

```

1   While (element number in UX > 1) do
2       Current_intersect_1 = UX [0]
3       Current_intersect_2 = LX [0]
4       Next_intersect_1 = UX [1]
5       Next_intersect_2 = LX [1]
6       Polygon_corners = [[Current_intersect_1, UY], [next_intersect_1, UY],
7       [next_intersect_2, LY], [Current_intersect_2, LY]]
8       Polygon_list.append(Shapely.polygon(Polygon_corners))
9       Remove UX [0] from UX
9       Remove LX [0] from LX
10  end while
11  return Polygon_list
End Polygon Forming in Vertical Orientation

```

---

In the case of horizontal orientation, the algorithm and concept to form the polygon for a horizontal camera angle are similar to the vertical one. We gather the intersection point between accepted Hough lines and the two left and right vertical boundaries as an example shown in Figure 4.15 (d). We gather the x value of all intersection points. Let's

represent a list of the y values of the intersection point in the left boundary with LeftY, where  $lefty_i$  is each y value contained inside LeftY:

$$LeftY = [lefty1, lefty2, lefty3, lefty4, \dots, leftyn] \quad (4.5)$$

let's represent a list of the y values of the intersection point in the right boundary with RightY, where  $righty_i$  is each y value contained inside RightY:

$$RightY = [righty1, righty2, righty3, righty4, \dots, rightyn] \quad (4.6)$$

Similar to the vertical orientation, the intersection x value of the horizontal orientation case will be constant numbers, represent by LeftX and RightX. The step to form the polygon is as shown in the pseudo-code in Algorithm 3.

---

**Algorithm 3.** Pseudo Code for Polygon Forming in Horizontal Orientation

---

**Input:** LeftX, RightX, LeftY, LeftY

**Output:** Polygon corners

Auxiliary Variables: Polygon\_list

Initialization: Polygon\_list = []

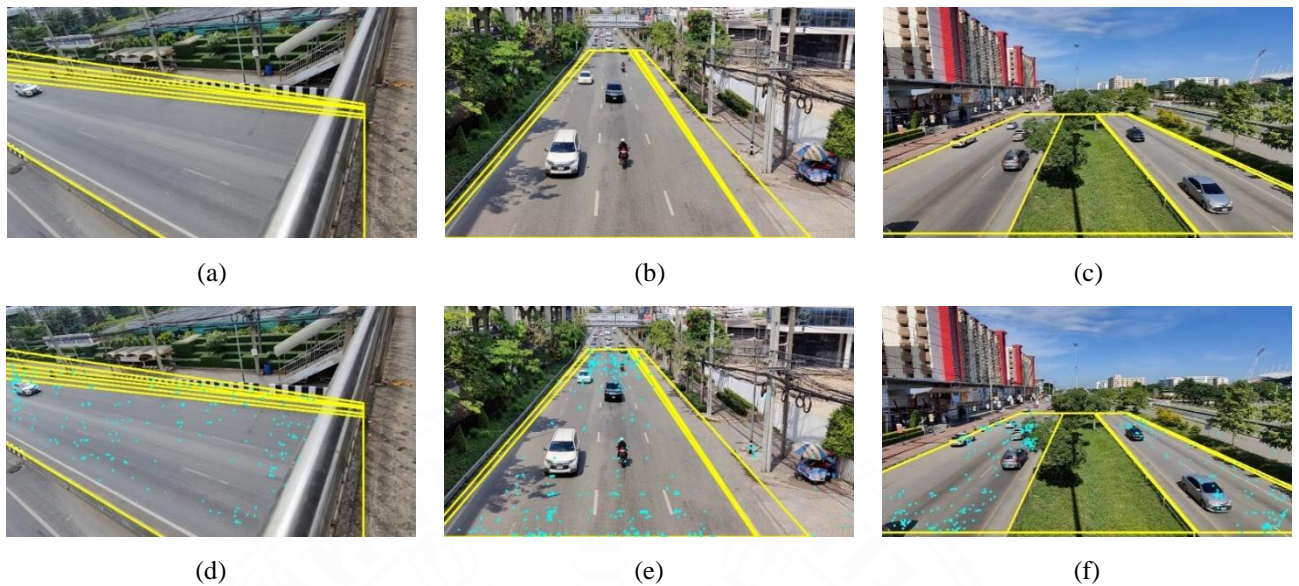
**Begin Polygon Forming in Horizontal Orientation**

```

1   While (element number in LeftY > 1) do
2       Cur_intersect_1 = LeftY [0]
3       Cur_intersect_2 = RightY [0]
4       Next_intersect_1 = LeftY [1]
5       Next_intersect_2 = RightY [1]
6       Polygon corners = [[XLeft, Cur_intersect_1], [XLeft, next_intersect_1], [XRight,
next_intersect_2], [XRight, Cur_intersect_2]]
7       Polygon_list.append(Shapely.polygon(Polygon corners))
8       Remove LeftY [0] from LeftY
9       Remove RightY [0] from RightY
10  end while
11  return Polygon_list
End Polygon Forming in Horizontal Orientation

```

---



**Figure 4.18** Form polygons (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes. Overlay start and stop point from vehicle information on the polygons (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes.

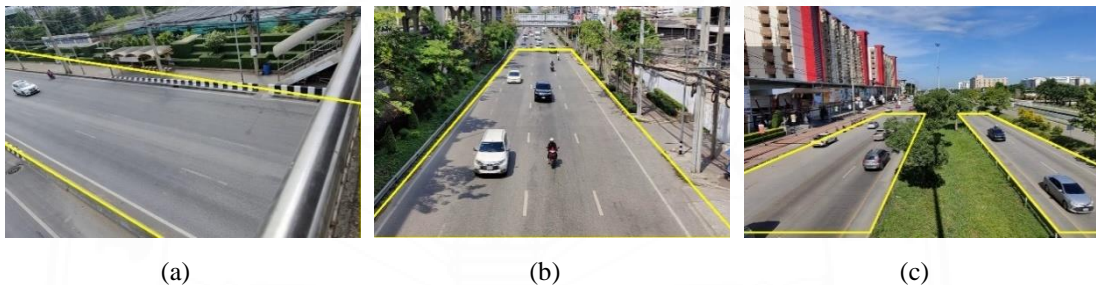
As we plot all the polygons we have received in this stage, the image will look like Figure 4.18 (a), (b), and (c) we can see that some polygons are not correctly referred to as the road lane. The method we use to choose which polygon is the road area is to overlay the vehicle information, start and stop point, onto the image like in Figure 4.18 (d), (e), and (f). It is obvious in Figure 4.18 (f) that some area does not have any start or stop points on, which is the grass area in the middle of the two roads. With this notice, we set the condition to select which polygon area to keep, and which one to remove. The number of points inside each polygon is set as the below equation.

$$Points = [num\_point1, num\_point2, num\_point3, \dots, num\_pointn] \quad (4.7)$$

Since we know the number of points inside each polygon, next we will check the number of points with the condition below. If the number of points is less than 20, which is the number we have tested with and proved that it is a number that is good with the vehicle information from a one-minute video.

$$\text{Accepted Polygons} = \begin{cases} \text{Add to accepted polygon list, } num\ point \geq 20 \\ \text{remove the polygon, } num\ point < 20 \end{cases} \quad (4.8)$$

After checking all polygons with this condition, we will have only polygons that represent the road areas. Sometimes, an area of the road is separated into two areas due to the detection of the Hough transform where it detects the centerline of the road. To fix this, we also set a condition to merge two areas that are next to each other to combine those two areas and correctly identify them as one area. As a final result of this step, the example result is shown in Figure 4.19 (a), (b), and (c).



**Figure 4.19** Final output (a) Horizontal orientation (b) Vertical orientation (c) Vertical orientation with two lanes.



## CHAPTER 5

### PROPOSED METHOD: WRONG-WAY DRIVING DETECTION

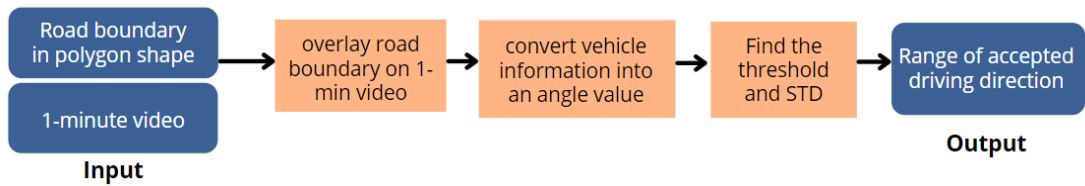
Many techniques have been introduced creatively to verify and detect wrong-way driving vehicles (Usmankhujaev, Baydadaev & Woo, 2020; Monteiro, Ribeiro, Marcos & Batista, 2007; Rahman, Ami & Ullah, 2020). However, there are some limitations from each paper that could be improved, for example, the vehicle detection in an occluded scenario, the manual validation of the correct moving direction, and multiple road lanes for wrong-way driving detection. In our proposed method, we aimed to improve the algorithm to overcome those challenges. The main concept of creating the wrong-way driving vehicle detection algorithm is to detect a vehicle that drives in the opposite direction from the majority direction on a specific lane. We based our assumption on the scenario where most of the vehicles are driving in the correct direction with a few law-breaking wrong-way driving vehicles. There are two proposed methods that we have introduced, where each of the methods has a different idea to detect vehicles driving in the minority direction.

#### 5.1 Majority-Based Correct Direction Detection

The first algorithm is Majority-Based Correct Direction Detection (MBCDD) where the algorithm will find the majority angle of all vehicles driving on a specific road lane and use that majority angle to draw a range of the accepted range of angle that would be considered as correct moving direction. There are two main parts to detecting wrong-way drivers using this algorithm: the validation part and the detection part.

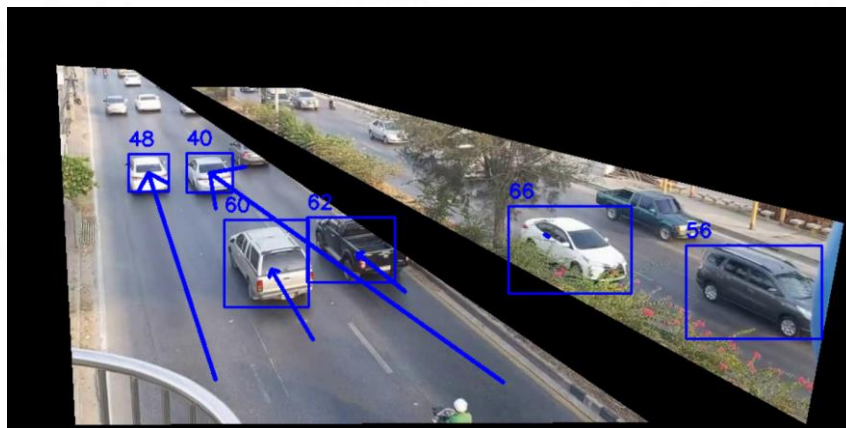
##### 5.1.1 Correct Direction Validation Part

To validate the correct direction in the MBCDD algorithm, we need to get the vehicle moving direction for each road lane. The method to obtain this information is what we have explained in Chapter 3.2.2. The overall system flow for the validation part in MBCDD is shown in Figure 5.1.



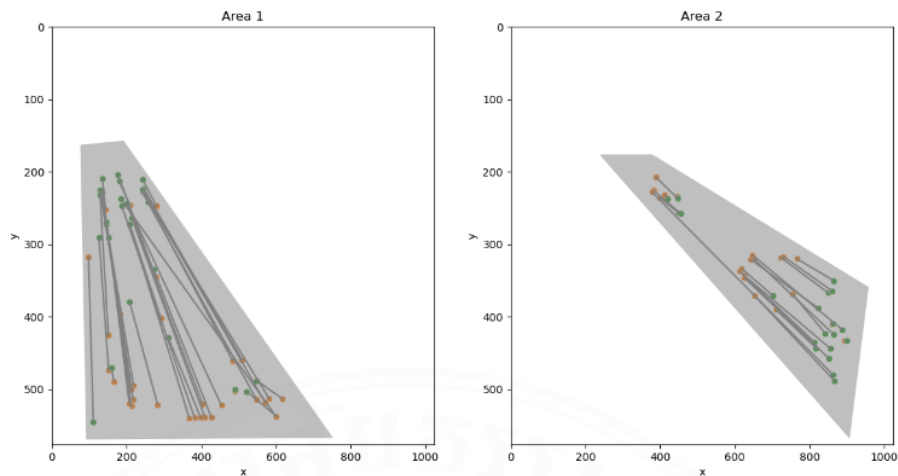
**Figure 5.1** System flow for validation part in MBCDD.

We begin with cropping the first one-minute video from the original input video. Then, we overlay the road lane boundaries that we have detected using the previous step algorithm, RLB-CCTV, onto all of the video frames. In this stage, the area outside the boundaries will be painted in black color, where we use the technique from OpenCV called “Bitwise mask”. Next, we detect and track all vehicles, including cars, motorcycles, and trucks, driving inside the boundaries while running the one-minute video. The example screenshot of the system while tracking and collecting each vehicle's information on the main road with a two-way direction is shown in Figure 5.2.



**Figure 5.2** Detecting and tracking all vehicles for direction validation.

After finishing the validation video, the system will show the total information that has been collected during the validation part as shown in Figure 5.3. The result shows two plots, where each plot presents one road boundary. The grey straight line inside each plot indicates the displacement of a vehicle driving direction from the start location, orange dots, to the stop location, green dots, as the video is playing.



**Figure 5.3** The direction of most vehicles for each lane of the road.

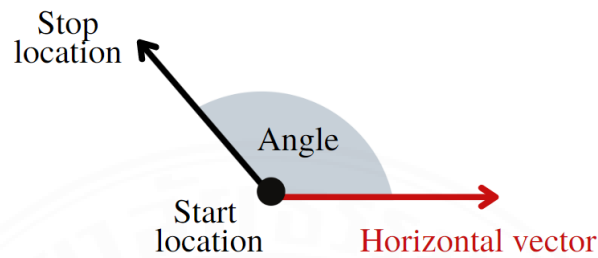
The five main values of each vehicle that the system collects are: Vehicle unique Identifier, start location, including x and y value, and stop location, including x and y value. The example piece of collected information is shown in Table 5.1.

**Table 5. 1** Information about each unique vehicle ID.

Vehicle ID	Start x	Start y	Stop x	Stop y	Direction Degree Angle	Angle divided by ten
16	509	459	246	223	138.097	13
4	280	247	242	219	135.764	13
88	283	521	208	379	117.842	11
41	403	520	201	244	126.2	12
79	98	317	111	545	-86.737	-8
75	571	518	243	224	138.129	13
55	491	502	489	500	135.0	13

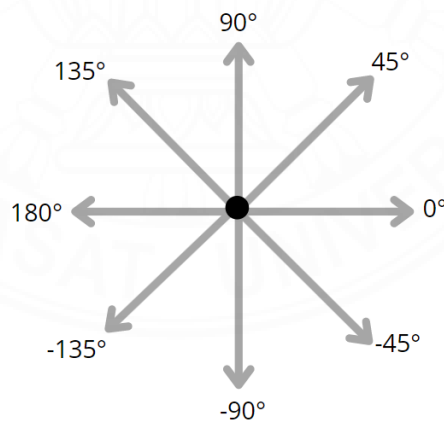
The total number of vehicles detected is seven vehicles and most of them are driving in the correct direction. To obtain the measurable value to identify the moving direction, we use the technique of angle calculation. We converted the start (Start x and Start y in Column 2 and Column 3) and stop (Stop x and Stop y in Column 4 and Column 5) location to be a vector. Then we draw another vector, where the start point

is shared with the first vector and the head of the arrow heading in the right direction with zero degrees as shown in Figure 5.4 in the red line. Next, we calculate and find the angle value between the two vectors.



**Figure 5.4** The method to find the angle value from the vehicle moving direction.

The possible value of an angle will range from -180 to 180 as shown in Figure 5.5 below. Assuming that a vehicle is driving vertically upward, the movement angle that the vehicle will be equivalent to is 90 degrees. Oppositely, if a vehicle is driving downward, the movement angle will be -90 degrees instead. With the huge different value between 90 and -90, we can conclude that these two vehicles are driving in the opposite direction from each other.



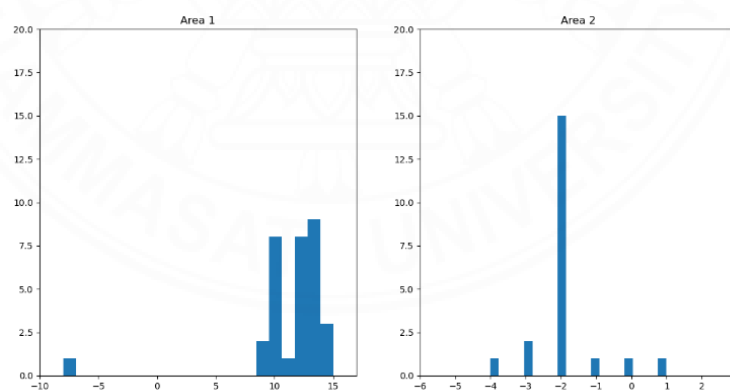
**Figure 5.5** All possible angle for the movement direction.

As we calculate all of the vector directions in Table 5.1, we will receive the result of the angle value in Column 6 in Table 5.1. To find the majority angle and obtain the result, we divided all of the angle values by ten, to easily find the mode value. From the final result in the last column of Table 5.1, we could see that most of the vehicle is driving in a similar direction except for vehicle ID 79, where its movement value is -8

while another vehicle ID has a movement value in the relatively similar value, range between 11 and 13. With this information, we can conclude that most vehicles are driving in the direction of 13, where we will set this value as the threshold. Moreover, the standard deviation value will be found with the below equation to determine the acceptable range of lowest value and highest value that the system will receive and verify as the moving correct direction. SD in the equation stands for Standard Deviation and the mean value is the average value of all the movement angles in Column 7 Table 5.1.

$$SD = \sqrt{\frac{\sum(\text{movement angle} - \text{Mean value})^2}{\text{Number of all vehicle}}} \quad (5.1)$$

After we propose the concept of angle movement value, threshold value, and standard deviation value, the histogram plot of all the movement directions in angle value we found in Figure 5.3 is shown in Figure 5.6 as the left plot is the first road lane and the right plot is the right road lane from Figure 5.2.



**Figure 5.6** Degree distribution for each lane of the road.

With this histogram plot for each road lane, we can see the high peaks on each plot where it indicates the most occurrence driving angle direction. We then select the highest peak value, or the statistic mode value, from each plot. As a result, the left plot's mode value is 14 and the right plot's mode value is -2. Moreover, the standard deviation value is found in the two plots as well, where the left plot's standard deviation value is

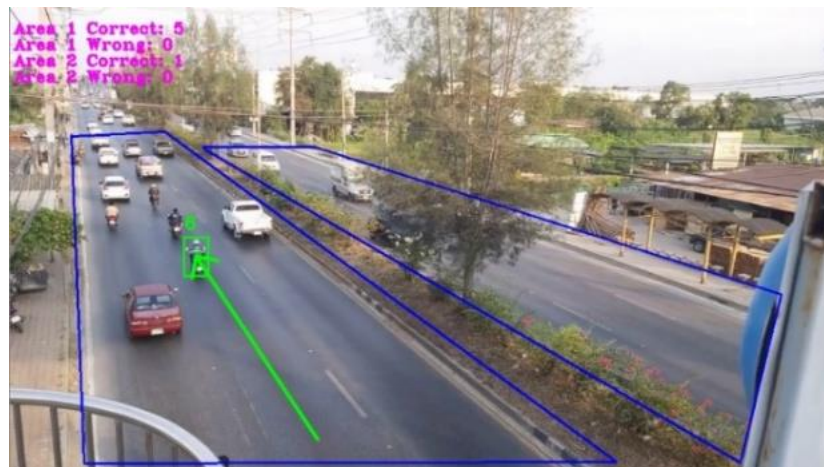
3.92 and the right plot is 1.11. The final output from this step will be the upper and lower bound of the vehicle moving direction that the system accepts as the correct moving direction, where the equation for upper and lower bound is shown in the equation below.

$$\begin{aligned} \text{Lower bound} &= \text{Threshold} - (SD * 2) \\ \text{Upper bound} &= \text{Threshold} + (SD * 2) \end{aligned} \quad (5.2)$$

From our example location, we can conclude that the upper and lower bound for the first road lane are 21.84 and 6.16 respectively. The upper and lower bound for the second road lane are -4.44 and 0.22 respectively. However, the upper bound for the first lane is 21.84, which exceeds the range of all possible angles that the system would produce, which is 18, or 180 degrees. In this case, we will set a condition to limit the possible value for upper bound and lower bound not to be greater than 18 and not to be lower than -18 respectively. If the value of the upper or the lower bound does not meet the condition, the highest and lowest value in the upper and lower bound will be set to 18 and -18 automatically.

### 5.1.2 Detection Part

After we found the insight information about the vehicle moving direction in each road lane according to our location example, next, we use the information, which is upper and lower bounds, to indicate the correct moving direction of each road lane. In this part, the video that will run is the video from the same location as the validation video but with a longer length. In this example, the testing video we use in the detection part has a length of 5 minutes. The screenshot of the video while the vehicles inside are being detected is shown in Figures 5.7 (a), (b), and (c), where the type of vehicle that the system is detecting is the motorcycle only because the motorcycle is the type of vehicle that is often found driving in the wrong direction in Thailand. However, the system can also detect other kinds of vehicles which are cars, buses, and big trucks. Figure 5.7 (a) shows the detection of motorcycles driving in the correct direction in the first road lane. The tail of the green direction arrow indicates the first location this motorcycle is detected in this road lane.



(a)



(b)



(c)

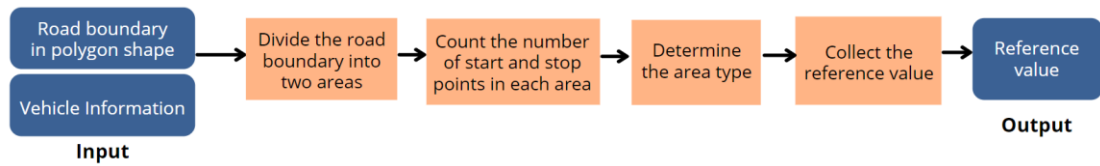
**Figure 5.7** Screenshot of the system detecting vehicles driving direction (a) Correct direction in the area one detected (b) Correct direction in the area two detected (c) Wrong direction in the area one detected.

To check with the condition of the upper and lower bound that we have found in the validation part, in the first lane, the correct driving vehicle should have an angle value within the upper and lower bound of 18 and 6.16. For the second lane, the correct driving direction is between the upper and lower bound of -4.44 and 0.22. The motorcycle detected in Figure 5.7 (a) has the moving direction's angle of 14, which is the value inside the upper and lower bound. For Figure 5.7 (b), the detected motorcycle inside the second road lane has a moving angle of -2. These two motorcycles from Figure 5.7 (a) and (b) are considered in the correct moving direction. However, in Figure 5.7 (c), the detected motorcycle inside the first road lane has the moving direction's angle of -8, which is not in the acceptable range we found from the validation step. Therefore, that motorcycle is being detected as a wrong-way moving vehicle.

## 5.2 Distance-Based Direction Detection

In our previous research (Suttiponpisarn, Charnsripinyo, Usanavasin & Nakahara, 2021) we have introduced the Majority-Based Correct Direction Detection (MBCDD) algorithm which is an algorithm that can validate the correct direction for each road lane, and detect the vehicle that drives in the wrong direction in that lane automatically. However, our previously proposed method, the MBCDD algorithm, still has some areas that need to be improved. MBCDD algorithm requires a one-minute cropped input video as input for validation, which is different from the DBDD algorithm which requires vehicle information. Even though the MBCDD algorithm is accurate and fast, in the process of direction validation on the edge device, it took twice as long time compared to the length of the input video. With this flaw, we tried to improve the algorithm and eventually come up with a more optimal algorithm that is more suitable to run on an embedded system. This newly proposed algorithm is called the Distance-Based Direction Detection (DBDD) algorithm. DBDD algorithm consists of two main parts: the validation part and the detection part, where the overall system flow is as shown in Figure 5.8.



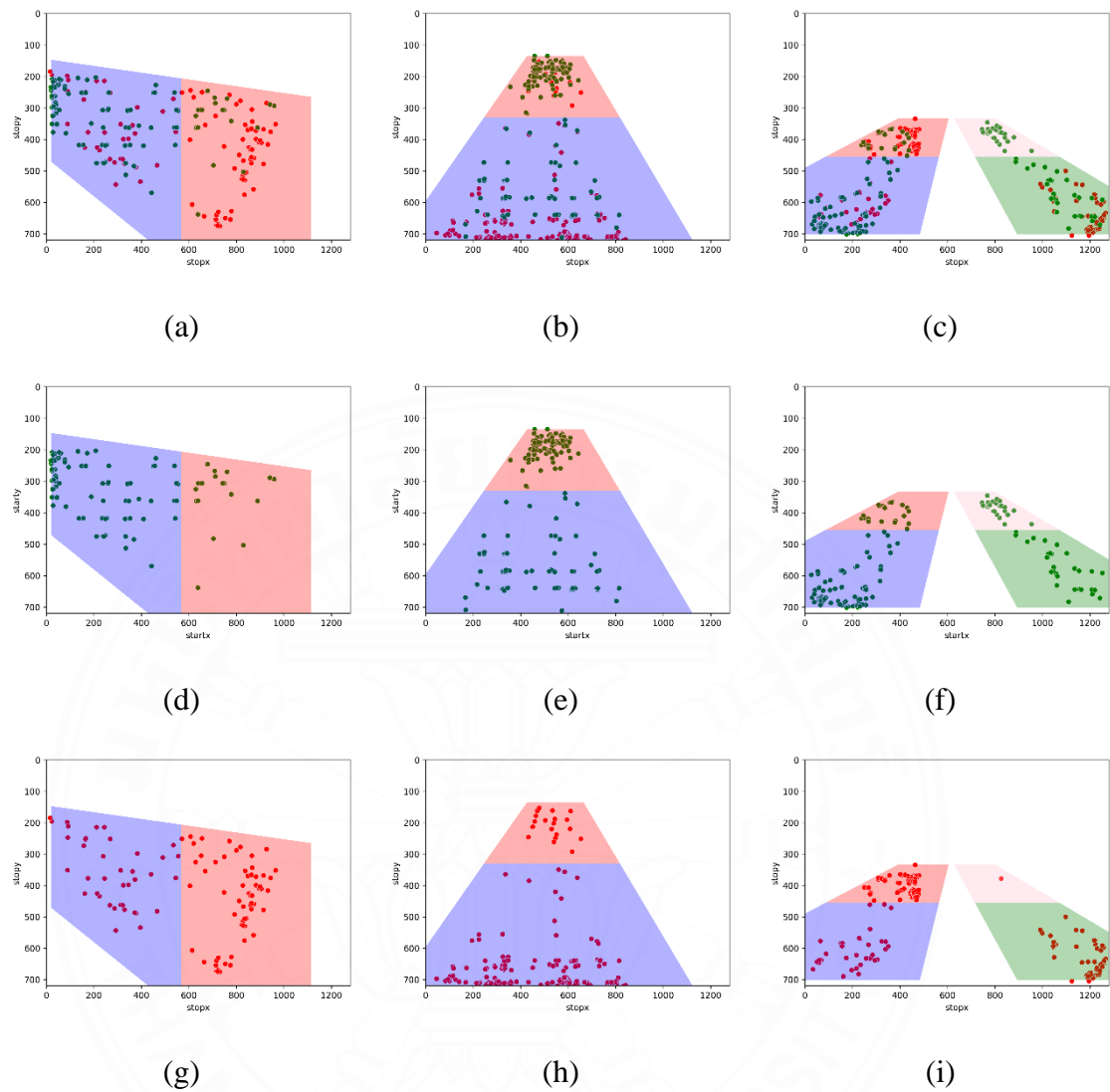


**Figure 5.8** System flow of DBDD algorithm.

The information that we have, supports us to come up with a straightforward algorithm to validate the correct direction of a straight road lane. In the DBDD algorithm, the required input is the detected road lane boundary from the RLB-CCTV algorithm and the vehicle information. There are 5 steps to perform this validation step of the DBDD algorithm.

### 5.2.1 Divide Road Area

To begin with, we divide the road lane area into two halves. If the orientation is vertical, the area will be divided into upper and lower areas. If the orientation is horizontal, the area will be divided into the left and the right areas. The second step is to overlay all the vehicle information onto the divided areas. As shown in Figures 5.9 (a), (b), and (c). The red dots are the start location of each vehicle, and the green dots are the stop location of each vehicle. With this information, we can roughly see that the start location, red points, of most vehicles, if the detection is fast enough and correctly, happens in the upper area of the road lane in Figure 5.9 (b). As well as the stop location of most vehicles tends to end in the lower area. To make the separate plot graph for clearer visualization, we plot Figures 5.9 (d), (e), and (f) to show only the start location of all vehicles in green dots, and in Figures 5.9 (g), (h), and (i) show only the stop location of all vehicle in red dots. After we have a clear distinction between the number of dots in each area, next, the third step is to decide on area type.

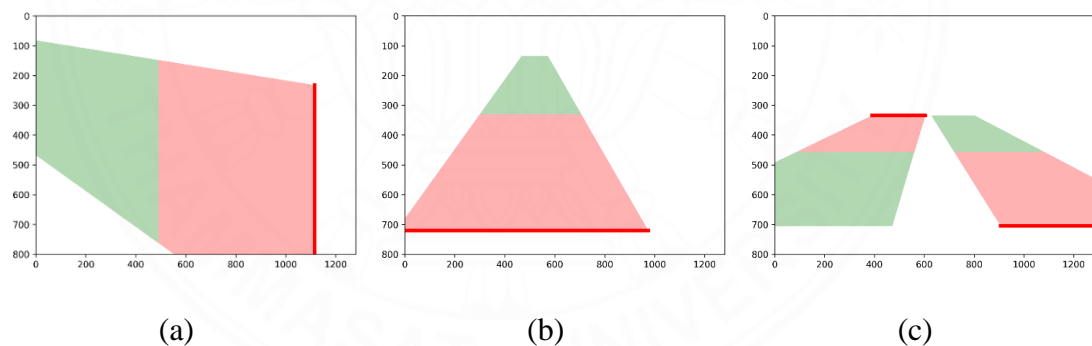


**Figure 5.9** Overlay all points from vehicle information on the areas (a) Horizontal orientation; (b) Vertical orientation (c) Vertical orientation with two lanes. Overlay start points from vehicle information on the areas (d) Horizontal orientation (e) Vertical orientation (f) Vertical orientation with two lanes. Overlay stop points from vehicle information on the areas (g) Horizontal orientation (h) Vertical orientation (i) Vertical orientation with two lanes.

### 5.2.2 Determine Area Type

We compare the number of dots that appear in each area for each of the start and stop locations. If the area has more start dots than the stop dots, then that area is the start area. If the area has more stop dots than the start dots, then that area is the stop

area. We color the start area with green color and the stop area with red color as shown in Figures 5.10 (a), (b), and (c). The fourth step of the validation part is to plot the reference stopping point. Assuming that a vehicle driving correctly in Figure 5.10 (b), the vehicle start location should start from the upper area, and come down to the stop area. This is the general moving behavior the vehicle should be like. In this step, we set an imaginary reference point on the stop area's side. The reference point only stores the y value from the cartesian coordinate system, where the y value is obtained from the location of the further edge of the stop area. This reference point will be used to check the distance between the vehicle's location and the reference point. The last step is to calculate the distance and determine if a vehicle is driving in the correct direction or wrong direction. The distance will be constantly calculated frame by frame as the video is running. The distance between the start location of each vehicle and the reference point will be kept as an initial distance value. As the vehicle is moving, the second location of the vehicle will be updated. The system will calculate the new distance value as well.

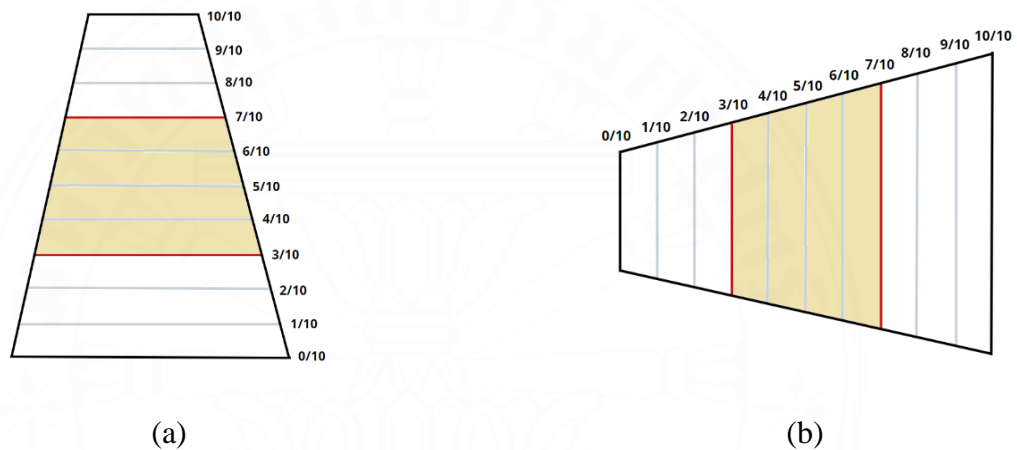


**Figure 5.10** Label the area and get the stopping reference point (a) Horizontal orientation with a stop area on the right (b) Vertical orientation with a stop area on the bottom (c) Vertical orientation with two lanes with opposite driving directions.

In this stage, the system will compare the old distance and the new distance. If the value decreases, it means that the vehicle is driving toward the reference point that is drawn on the stop area side, or the vehicle is driving in the correct direction. However, if the value increases, it means that the vehicle is driving away from the reference point, or the vehicle is driving in the wrong direction. In the third frame of the video, the new distance value is calculated. The system will compare this distance value with the first

initial distance value. As a result, this should confirm the moving direction that the vehicle is driving toward.

However, there is an observation about the reference point that instead of marking the reference point on the stop location, it can be marked on the start location as well. These choices of setting for the algorithm do not make an important difference in results. The reference point can refer to the start location as well, but the condition to check the moving vehicle should be changed accordingly as to when the vehicle's moving distance is increasing, it would mean it is driving in the correct direction.

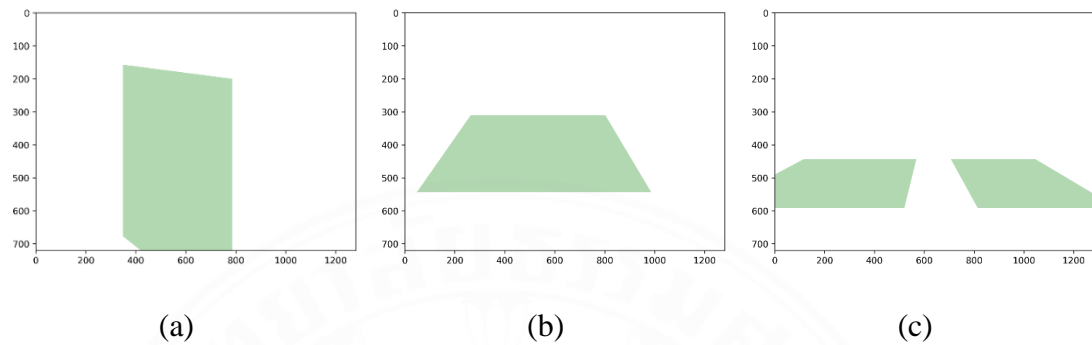


**Figure 5.11** Length of detected road boundary from the RLB-CCTV algorithm (a) Vertical orientation (b) Horizontal orientation.

With the concept of the reference point, we can observe the moving direction of each vehicle that drives inside the boundary. Moreover, we have considered the case where the lane boundary could be shorter in length for the shorter computing time of the system and a higher accuracy rate. The system is designed to track all vehicles inside the boundary and calculate the distance between each vehicle and the reference point. If the length of the road is long, the calculation on each vehicle is longer too. As we use YOLOv4 Tiny to detect the vehicles on the road. We focused on detecting the vehicle with visible size. Some far vehicles would confuse the detector. For the two stated reasons. We cropped the length of the road boundary by dividing the road into 10 ranges as shown in Figures 5.11 (a) and (b).

In the range of 3/10 to 7/10 will be the focused section of the DBDD algorithm. We chose this range of sections due to the result when we run the sample video on our

proposed system. We remove the range 0/10 to 3/10 because as the vehicle approach closer to the camera in the vertical orientation, the discontinue tracking is likely to happen.



**Figure 5.12** The road boundaries are cropped for a range of 3/10 to 7/10

(a) Horizontal orientation (b) Vertical orientation

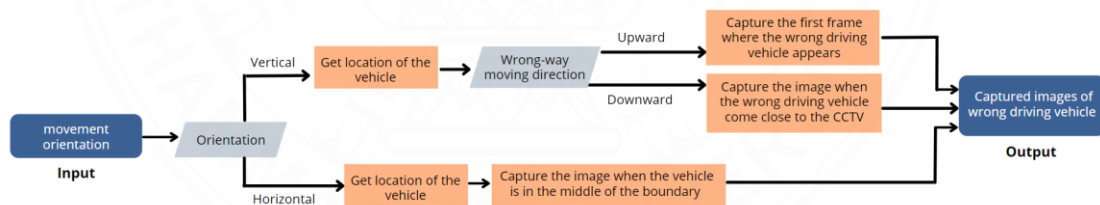
(c) Vertical orientation with two lanes.

We removed the range 7/10 to 10/10 because the far objects are hardly detected. The range we remain is adequate for the system to detect each vehicle by the time they are moving, and objects are not too close nor too far. For the horizontal orientation, in Figure 5.11 (b), the purpose for cropping the range is solely because of less computation time. As a result, the road boundaries from Figure 5.10 are cropped as shown in Figure 5.12.

## CHAPTER 6

### INSIDE BOUNDARY IMAGE CAPTURING FEATURE

After detecting the vehicles that drive in the wrong direction, the DBDD algorithm will return the summary report counting the total number of vehicles driving in each direction. Apart from the summary number, we would like to collect the information output in the form of images, where the system will capture the image of the wrong driver in the act. The challenging part of this algorithm is the high-speed movement of the vehicles on road, where we have to find the right moment to capture the clearest and closest shot of that driving vehicle. The proposed algorithm we designed for this task is called the Inside Boundary image capturing feature (IBI Capture). The benefit of this image capturing algorithm is to identify the individual driver appearance who drives in the wrong direction and to trace back the evidence of wrong driving behavior. However, in this research, we do not further use any personal information of the driver. The experiment of our research is just to test the system's accuracy and clearness only. The system flow for IBI capturing feature is shown in Figure 6.1.



**Figure 6.1** System flow of IBI capturing feature.

There are two moving orientations of the road depending on the location and camera angle. To create the IBI capturing feature, we need to consider both orientations. The flowchart in Figure 6.1 will help determine the steps and decisions of how our proposed IBI capturing feature will perform on different kinds of road angle orientation.

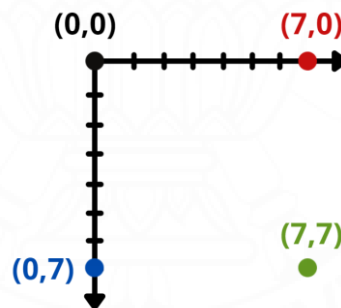
#### 6.1 Vertical Orientation Image Capturing

In the case that road orientation is vertical, the vehicle can drive upward and downward. The moment that we can see the vehicle come closest to us is when the

vehicle is start closest to the lower boundary or the CCTV camera. With this idea, we try to capture images of the vehicles that drive in the wrong direction with a condition where the vehicle comes close to the lower boundary as much as possible.

There will be two cases where we are concerned about the vertical orientation. First, when the correct driving direction is to drive upward, the wrong driving vehicle will drive downward. The closest moment to capturing the wrong driving vehicle is when the vehicle is about to leave the lower boundary. Second, if the correct driving direction is to drive downward, the wrong driving vehicle will drive upward. The closest moment to capturing the wrong driving vehicle is when the vehicle first appears into the lower boundary. With these two different cases, we apply different methods to capture the image.

When working with images on the OpenCV library, the coordinate system will be arranged in the form of an image coordinate system where the y-axis value will be flipped from the normal coordinate, as shown in Figure 6.2.



**Figure 6.2** Image coordinate system.

The method to determine if the wrong-way driving vehicle is going to drive upward or downward is to check with the height of the stopping area reference value and the height of the lower boundary as shown in the equation below.

*Wrong way driving direction*

$$= \begin{cases} \text{upward,} & \text{height of reference point} = \text{height of lower boundary} \\ \text{downward,} & \text{height of reference point} > \text{height of lower boundary} \end{cases} \quad (6.1)$$

To find the closest and clearest moment to capture the images, the system will require the y value of the current location of the vehicle, represented with CY, and the y value of the lower boundary, represented with YL. If we are considering the case where the wrong way driving vehicle will drive downward, we will calculate the difference value between CY and YL. The moment that the system will capture the image is when the difference value of CY and YL is as close as zeros. If the difference value is zero or lower, it means that the lower part of the vehicle bounding box is laying on or below the lower boundary line. In that case, we would not get a whole component of the vehicles. As an example, shown in Figure 6.3, the motorcycle's front wheel is being blocked by the road lane boundary. Therefore, we set a value of 20 pixels to be the maximum difference value between the CY and YL points to make a screenshot image of the vehicle. The value of 20 pixels has been evaluated to be a proper value to leave some distance between the vehicle's bounding box and the YL as examples of captured images are shown in Figure 6.4.



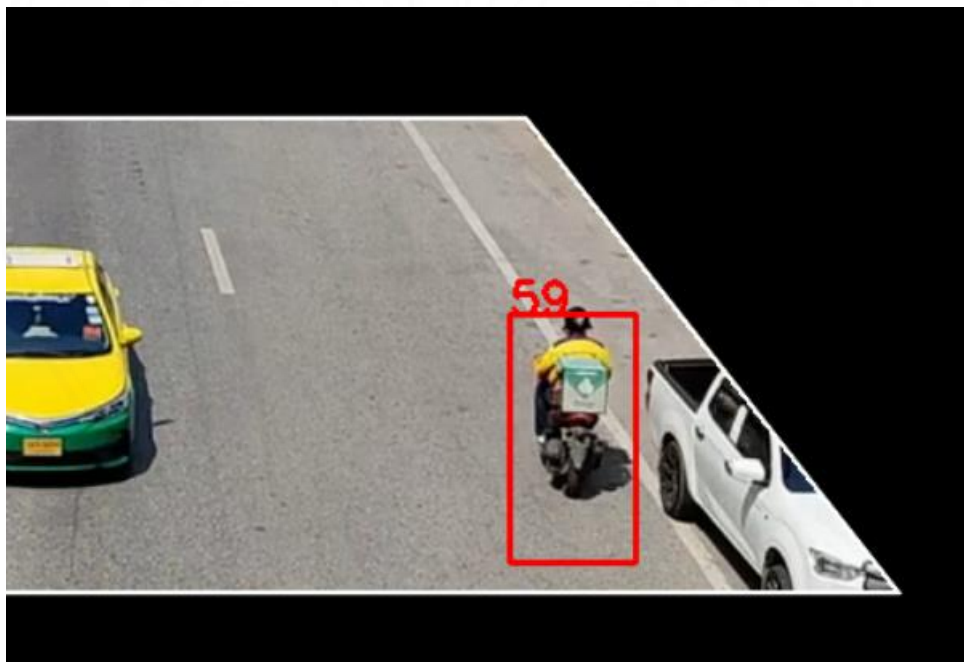
**Figure 6.3** The screenshots when the CY bounding box is lower than YL.





**Figure 6.4** The screenshots when the CY bounding box is above YL by 20 pixels.

For the case where the wrong way driving vehicle is driving upward, the system will capture the first frame where the vehicle appears and detected it as wrong-way driving as an example screenshot is shown in Figure 6.5.



**Figure 6.5** The screenshots when the wrong-way driving vehicle is driving upward.

## 6.2 Horizontal Orientation Image Capturing

The image capturing in horizontal orientation is a bit trickier on how to find the closest moment where the vehicle is closest to the camera. However, the method we proposed is to select the frame where the bounding box of detection is the biggest. The

example screenshot of the vehicle driving in the wrong direction in horizontal orientation is shown in Figure 6.6.



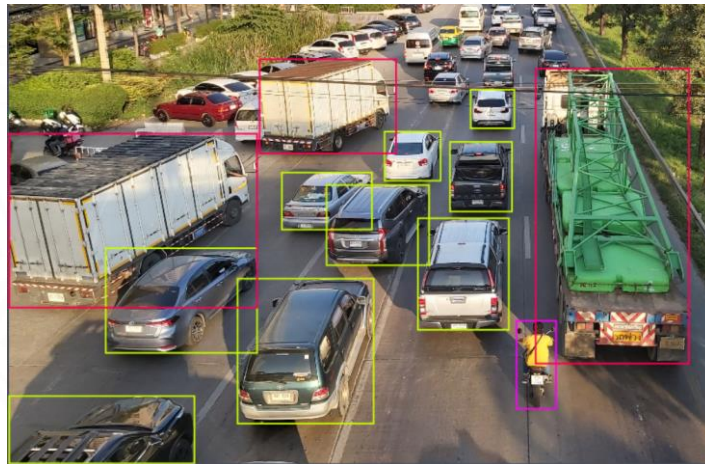
**Figure 6.6** The screenshots when capturing vehicles driving in the wrong direction in the vertical oriented video.

## CHAPTER 7

### EXPERIMENTAL RESULTS

#### 7.1 Dataset

The image dataset we collected came from various locations and times. The location where we capture the images is on several skywalks around Bangkok and Pathum Thani province using our smartphones to capture the images. In most locations, we can see lots of cars and motorcycles. Some area also has a lot of large vehicles such as truck and bus. The example image we gather is shown in Figure 7.1. We gathered a total of 437 images. To detect the various kind of vehicles on the road, we annotated the objects into three classes: Motorcycles, cars, and large vehicles. The majority of the camera angle we gathered in the dataset is vertical orientation, which is the front view of the vehicles.



**Figure 7.1** Labeling the dataset with three classes: Motorcycle, car, and large vehicle.

#### 7.2 Vehicle Detection

As we have prepared the dataset and the label, we then train the model with YOLOv4-Tiny. The portion of the train set and validation set is in the ratio of 80 to 20 where the augmentation of the dataset is included. The total image for training is 1068 images. The result and accuracy of the model after training for motorcycles, cars, and large vehicles are as shown in Table 7.1.

**Table 7.1** Accuracy of the YOLOv4 Tiny model.

Parameters	Class (percent)			mAP (percent)
	Motorcycle	Car	Big vehicle	
Results	71.89	78.0	86.25	78.71

With the requirement to apply the vehicle detector on an embedded system, we choose to use YOLOv4 Tiny as it is fast and consumes less time to detect. However, YOLOv4 Tiny has a trade-off with accuracy. Song, Liang, Li, Dai, and Yun (2019) train a vehicle detection model on YOLOv3 with more than 10,000 images of the dataset. With more than ten times greater number of datasets, their accuracy achieved up to 87.88% for mAP. To compare the result with other research that implements YOLOv4 Tiny onto the pedestrian detection system, Roszyk, Nowicki, and Skrzypczyński (2022) show that the average accuracy is 0.557 for mAP50. With this observation, if we would like to implement a small detector that can run on an embedded system at a real-time speed, YOLOv4 Tiny would perform that task perfectly but it might not achieve high accuracy as YOLOv4 or YOLOv3. However, our trained model is accurate compared to other results and can detect the target objects in our framework.

### 7.3 Framework's Result

#### 7.3.1 Road Lane Boundary Detection based on CCTV Algorithm

After collecting several videos from many locations, we test the videos with our proposed algorithm to detect the road lane boundary. Some video contains one road, and some contain two roads. According to the result that we have generated, all of them have a satisfactory result where it has achieved their task to detect the road lane. Some of the locations we have gathered and the lane boundary detection result is as shown in Figure 7.2.



**Figure 7.2** Output images with road boundaries using the RLB-CCTV algorithm.

### 7.3.2 Distance-Based Direction Detection Algorithm

In the DBDD algorithm, there are two steps to proceed: validation and detection steps. First, we show the result of the system after validating the correct moving direction by determining the start area and the stop area from the road lane boundary we obtained from RLB-CCTV. As we had described, DBDD will divide the road boundary into two areas. If the road boundary's orientation is vertical, the boundary will be split into the upper and lower area. If the road boundary is horizontal, the boundary will be split into a left area and a right area. Table 7.2 shows the result from the DBDD algorithm in the validation step, where it shows the information on the number of the start and stop points in each area and determines the area type compared to the ground truth.

**Table 7.2** The number of start and stop points on each divided road area.

Video	Length (minute)	Area	Number of Points		Area Type	
			Start	Stop	Prediction	Ground Truth
1	2:12	Area 1	0	95	Stop	Stop
		Area 2	50	35	Start	Start

		Area 3	23	2	Start	Start
		Area 4	2	53	Stop	Stop
2	3:01	Area 1	0	28	Stop	Stop
		Area 2	18	1	Start	Start
		Area 3	21	0	Start	Start
		Area 4	0	23	Stop	Stop
3	5:00	Area 1	4	119	Stop	Stop
		Area 2	68	40	Start	Start
4	2:31	Area 1	2	64	Stop	Stop
		Area 2	46	31	Start	Start
5	2:59	Area 1	111	37	Start	Start
		Area 2	2	194	Stop	Stop
6	2:04	Area 1	100	35	Start	Start
		Area 2	4	228	Stop	Stop
7	4:07	Area 1	72	34	Start	Start
		Area 2	0	151	Stop	Stop
8	2:00	Area 1	3	88	Stop	Stop
		Area 2	43	35	Start	Start
9	2:03	Area 1	0	71	Stop	Stop
		Area 2	50	16	Start	Start
10	2:02	Area 1	3	41	Stop	Stop
		Area 2	105	21	Start	Start

From Table 7.2, we can see that all areas from road boundaries were determined correctly using the DBDD algorithm in the validation part. The sample location we use for testing our algorithm is a mixture of vertical and horizontal camera orientation. Some location has two-lane and some have one lane. This result shows that our algorithm can handle verifying the start and stop areas accurately. After we have done the validation step, next is the detection step. In the detection step, the system will detect the wrong-way driving vehicle by calculating the distance between the vehicle and the edge of the stop area boundary. If the distance keeps

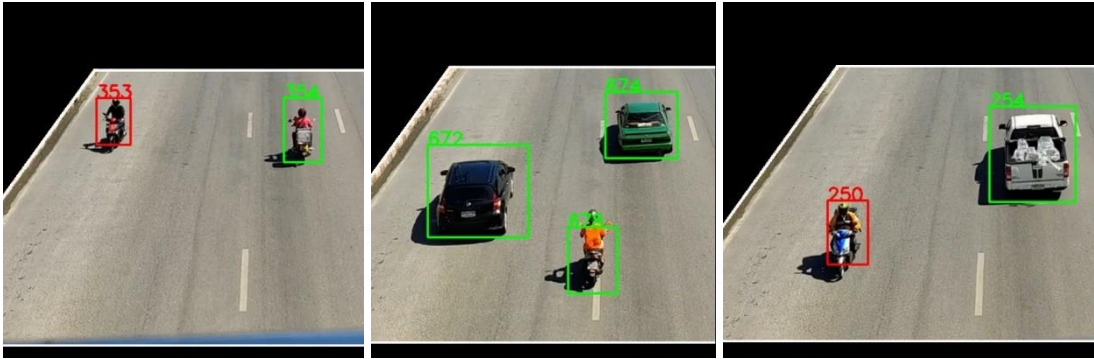
decreasing in each frame as the vehicles are moving, then, those vehicles are driving in the correct direction. The result of our algorithm that has been tested with videos from ten locations is shown in Table 7.3.

**Table 7.3** Numbers of motorcycles were detected as wrong-way and correct-way driving using the DBDD algorithm.

Video	Lane	Ground truth		Computer		Embedded system		Speed (FPS)
		Correct driving	Wrong driving	Correct driving	Wrong driving	Correct driving	Wrong driving	
1	1	9	1	9	1	9	1	22
	2	6	0	5	0	6	1	
2	1	8	0	9	0	7	0	23
	2	3	0	2	0	3	0	
3	1	51	15	51	15	49	17	27
4	1	20	6	19	6	19	6	27
5	1	44	4	46	5	47	3	18
6	1	26	5	25	5	26	5	19
7	1	70	1	68	1	68	1	26
8	1	25	10	26	11	28	10	25
9	1	32	2	29	2	33	2	32
10	1	13	5	14	6	12	6	29

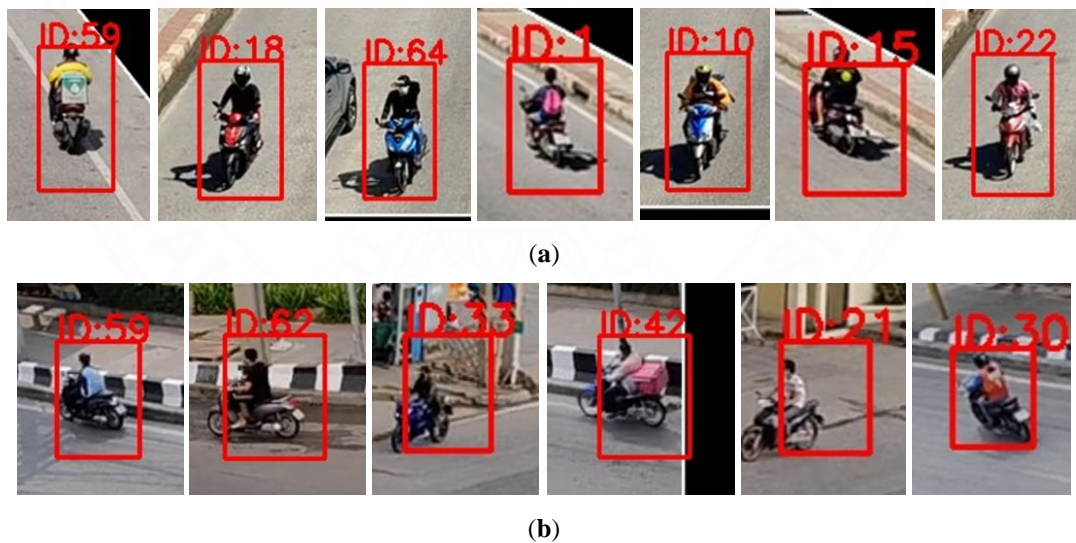
We have tested our system on a normal computer and on an embedded system (Jetson Nano). The results of detection are quite similar and close to the ground truth. There is some slight error value of detection due to the accuracy of object detection and tracking of our system. To evaluate the accuracy of our proposed system with these ten videos, the accuracy of our system is 95.225% when running on a personal computer. The accuracy our system achieved when running on Jetson Nano is slightly lower due to limited resources for object detection and tracking, which is 94.663%. The average FPS when running the detection on Jetson Nano is around 24 FPS, which is suitable for real-time detection. Figure 7.3 shows the screenshot of the detection process while the

system is detecting the wrong-way driving vehicle on a video with vertical orientation. The system can track and detect vehicles accurately.



**Figure 7.3** Screenshot of the DBDD algorithm while detecting vehicles driving in the correct and wrong direction.

### 7.3.3 Inside Boundary Image Capturing Feature



**Figure 7.4** Output images of the wrong-way driving vehicles using the IBI capturing feature (a) Vertical orientation road (b) Horizontal orientation road.

To capture the clearest image, IBI capturing feature can capture the closest moment where the vehicle comes closest to the camera. As a result, shown in Figure 7.4 represent the screenshot image that has been cropped to the size of the vehicle that drives in the wrong direction. Suppose we have a high-resolution CCTV or camera; the screenshots might be able to bring out all the detail of the law-breaking driver and the



vehicle they are using. However, since we were collecting the sample videos with the smartphone, the resolution is limited and this feature is the ideal concept of what the framework can do. Figure 7.4 (a) are the example captured images with IBI capturing feature when vehicles are driving in the vertical orientation. For the video that has the horizontal orientation, the example captured images from the IBI capturing feature are shown in Figure 7.4 (b).



## CHAPTER 8

### CONCLUSION AND DISCUSSION

In this thesis, we have introduced our proposed approaches that are used for wrong-way driving vehicle detection with the additional features of road lane detection algorithm and evidence capturing for wrong-way drivers. As a final result, we combined all algorithms and features into one pipeline of algorithms execution and call it, the “WrongWay-LVDC Framework”. However, there are some limitations and challenges that have been raised during the framework development and we would like to mention them in this Chapter.

#### 8.1 Limitations

The scope of our research is to detect the road lane with a straight lane line and be able to implement the system onto an embedded system, such as a Jetson Nano, with a real-time running speed. Therefore, there are three main limitations to our system we would like to identify

The first limitation is running the system on Jetson Nano. Jetson Nano is an effective embedded system that can perform image processing tasks. However, to achieve the high number of FPS, there are several circumstances we have to adjust that would be most suitable for the embedded system, such as, do not show the detecting video, and narrow the road boundary to be smaller. If the boundary is at its original size, it would take more computing time to compute a large area. Therefore, we decided to reduce the size of the road boundary after detecting it with the Road-CCTV algorithm.

The second limitation is the various road condition. It is challenging to accept all types of roads to process in our system. However, there are several factors that would make the system become inaccurate and get confused with the overload data in the RLB-CCTV algorithm and DBDD algorithm. If we do not set a condition where the eligible video of the road must not have the side road blocking the road line, the RLB-CCTV algorithm will not be able to detect the side road as it should. This is an interesting challenge and needs deeper research on how to detect the side road even if there are objects blocking it. Another case is the curve line road. Our system could not

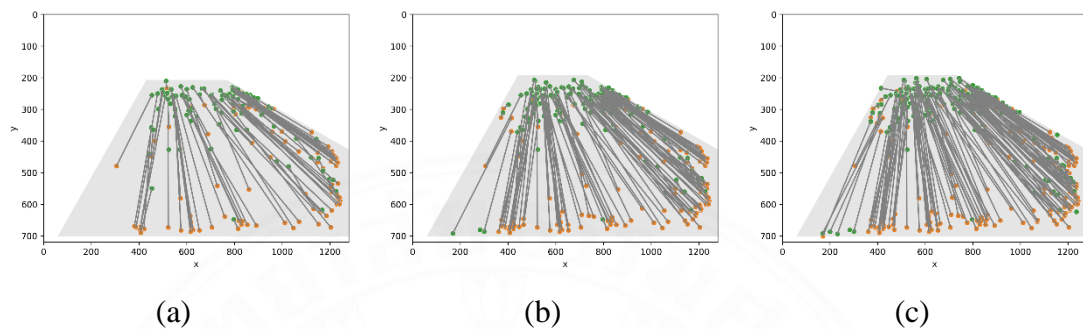
function properly if the road is not straight because the DBDD algorithm is designed for either horizontal or vertical orientation. To determine the orientation of the curved road, it would not be functional and need a more advanced algorithm to handle such situations.

The last limitation is the capability to validate the correct driving direction in terms of speed and accuracy for MBCDD, which leads to the replacement of the proposed wrong-way driving vehicle detection algorithms from MBCDD to DBDD. In our WrongWay-LVDC framework, we use DBDD as the algorithm to detect the vehicle that drives in the wrong direction. According to our experimental results, MBCDD serves a great accuracy and performance in vehicle wrong-way detection as well. However, there are some aspects in the validation step that MBCDD remains at a weak point. Generally, we would like to spend a shorter period of time validating the correct direction on a road boundary with great accuracy. Therefore, the length of the validating video should be short but the challenging part is, can a short validating video produce a threshold value with enough accuracy especially the validation on an embedded system. We have tested this idea by trimming the video length to be 1,000 frames length, 2,000 frames length, and 3,000 frames length. In Table 8.1, we test MBDCD with the stated configuration of the video's length for the validation step on Jetson Nano.

**Table 8.1** MBCDD algorithm in the validation step on Jetson Nano.

Video Length (Frame)	Validation				Detection		
	STD	Threshold	Time taken (Seconds)	Speed (FPS)	Correct-way driving vehicle	Wrong-way driving vehicle	Speed (FPS)
1,000	6.053	0	174	7	51	8	23
2,000	6.130	7	303	7	52	12	20
3,000	6.498	14	450	7	52	15	20

From the result in Table 8.1, threshold values are increasing according to the number of frame lengths. Figure 8.1 shows the detected vehicle movements that are used in the validation step. The higher number of frames will increase the accuracy of the majority direction most vehicles are moving toward.



**Figure 8.1** Detected vehicle movements (a) 1,000 frame length (b) 2,000 frame length (c) 3,000 frame length.

The detection result from Table 8.1 shows a varying number. However, the ground truth of vehicles moving in the correct and wrong direction in this sample road area is 51 for correct moving vehicles and 15 for wrong-way driving vehicles. To compare the error, frame number at 1,000 produces 7 miscount vehicles, frame number at 2,000 produces 4 miscount vehicles, and frame number at 3,000 produces 1 miscount vehicle. With this information, we can tell that the minimum number of frame length to use in the validation step for MBCDD running on Jetson Nano is 3,000 frames, which takes around 7.5 minutes to validate the direction while the actual video run times for 3,000 frames should be only around 1 minute and 40 seconds. For DBDD, the validation step is simpler where the video frame length at 1,000 frames already produces an accurate validation result. With these concerns, we decided to use our proposed DBDD algorithm to be a part of our WrongWay-LVDC framework.

## 8.2 Challenging Constraints

To test our proposed algorithms and whether they would handle various types of scenarios and constraints, we have gathered different possible scenarios that would happen while we run the framework. As a result, there are four cases of challenging constraints that the framework may face and the method that the system would handle those situations.

The first case to consider is the resolution of the general CCTV cameras. As we could not access the CCTV camera for the video, in this research, we imitate the CCTV camera angle and recorded the video using the smartphone. The video we have received from the smartphone is in the resolution of 1920 pixels in width and 1080 pixels in height. We converted the resolution to 1280 pixels in width and 720 pixels in height for faster processing. As we checked the resolution of CCTV, the converted resolution of the input video is equivalent to 720p HD video from CCTV. There are many several higher resolutions of video that are collected from CCTV. However, as long as the resolution of the input video from the actual CCTV is greater than 1280 x 720 pixels, the input video should be compatible with our system.

The second case is when the system meets the heavy traffic scenario. During rush hour, we can often face bad traffic, where all vehicles on the road move slowly or do not move at all for a certain time. As the system's default requires one minute video for the validation part, in the case of traffic jams, the distances of each moving vehicle in the video might be too short and the system might not get the significant information to validate the correct direction of a road lane. In the case that the vehicle is moving slowly for the video length of one minute, for the MBCDD algorithm, this case would not affect the efficiency of the system to validate the correct moving direction because even the tiny direction movement all the vehicles produce, it is enough for the system to understand the majority moving direction of all vehicle. However, this will severely affect the validation part of the DBDD algorithm since the algorithm will divide the road into two areas, the start and stop area. If the vehicles do not have enough time, within one minute, to drive across from the start area to the stop area, the system will fail to validate the correct direction. However, this situation can be solved directly by increasing the time for the validation video to be longer, depending on the time of the day of the video that is selected to be the validation video. The system should avoid using the video during rush hour to validate the correct direction. If it is unavoidable, the longer length of the video, such as 3 or 5 minutes, should be used to validate instead, as this will be a solution for the system to obtain more accuracy in direction validation.

The third case is when there is a vehicle that is driving too fast. Oppositely to the previous case, we also consider the capability of the system to detect fast-moving

vehicles. The collected video has a resolution of 30 frames per second. In the case that the vehicle is driving too fast and it appears in only one frame in the video, the system will not be able to track the moving direction due to the low information provided. The possibility of detection is when a fast-moving vehicle appears for at least two frames inside a video. In this case, there will be two challenges for our proposed systems: tracking capability and direction specification.

FastMOT tracking algorithm can track multiple vehicles at the same time and have a high ability to continuously track an object that moves fast. As we tested, FastMOT can be adjusted the configuration and is reliable for object tracking. In the configuration parameter inside the mot.json file in the FastMOT algorithm, some parameters will support the tracking of fast-moving vehicles without discontinuous tracking. The value of parameters we have adjusted to support this function is shown in Table 8.1

**Table 8.2** Adjusted parameters in mot.json file.

Parameter	Adjusted Value	Description
Detector_frame_skip	1	Track object on every one frame
Conf_thresh	0.5	the higher, the refiner the detection will get.
Max_reid_cost	0.6	the higher, the fewer number of tracking
Iou_thresh	0.1	the tracking is not disconnected

For direction validation, this might be a challenge to detect the moving direction of a vehicle that appears inside the video for two frames. However, even a small number of the frame that a vehicle appears in, our algorithms can detect and specify the moving direction. Both of our wrong-way driving detectors, MBCDD, and DBDD, only require the two locations of a vehicle to determine the distance and moving angle of a vehicle. With this idea, we can conclude that even if a vehicle is moving very fast, with only two frames it appears, our algorithms can verify the moving direction of that vehicle.

As we tested our system and get the experimental results, we tested our system with short video clips of around 3 to 5 minutes. However, we are also concerned in the

case where the input video would take a longer length as the ideal length of around one hour per video clip, where the performance of detection will be affected. Therefore, the fourth challenging case is the long length of the video input. As we run the wrong-way driver detection on PC, the speed is up to 80 FPS in a normal traffic scenario. With the high performance on PC, the trend of detecting speed will not drop below the real-time speed, which is around 30 FPS, even if the input video is very long. However, running a long video on small embedded is a major concern because embedded systems have limited resources, and always gets easier to get a high temperature and decrease the performance of detection. As we tested our system on Jetson Nano, the average speed is around 24 FPS. In the scenario where the traffic is heavy on the testing video, the speed may drop below 20 FPS. Therefore, there will be many factors that would affect the speed of wrong-way driver detection on an embedded system. There are potential solutions that support the wrong-way driver detection. First, keep the temperature of the embedded system to be low and avoid the overheat condition while detecting. Second, skip the frame of detection for the input video. If the device detects all vehicles every frame, it will consume excessive energy and would slow down the process. Lastly, turned off the interface display while running the system, to reduce the power consumption. With these approaches, the detection on embedded systems would remain in a good performance detecting long video clips.

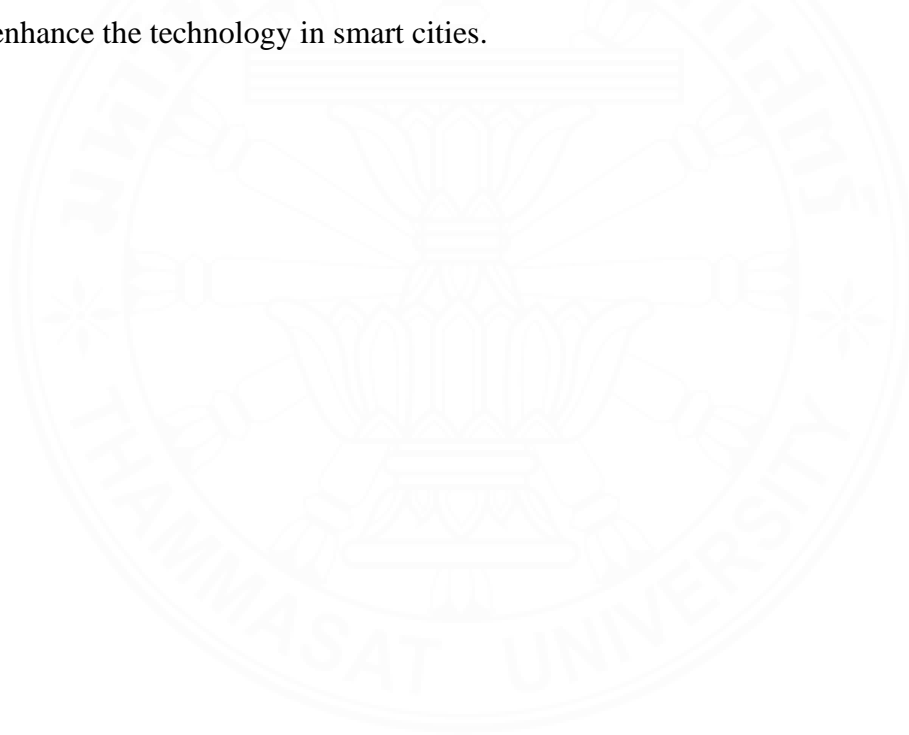
### **8.3 Summary of Thesis**

We have proposed the WrongWay-LVDC framework, which is designed to perform the three main tasks: Lane detection using the RLB-CCTV algorithm, Validating and Detecting wrong-way driving vehicles using the DBDD algorithm, and Capturing evidence images of the wrong-way vehicles using IBI capturing feature. Each of these tasks will run in a seamless pipeline flow manner automatically, which is convenient and functional to use in a broad traffic area. Furthermore, our system has proven for being able to implement in an embedded system at a real-time speed of 24 FPS on average. The accuracy of our framework is 95.23% while running on a personal computer and 94.66% while running on an embedded system.

During our experiment to develop this framework, we proposed an MBCDD algorithm for driving direction validation and wrong-way driving detection. The result

of the MBCDD algorithm produces a high accuracy. However, as we try running this algorithm on an embedded system in the validation part, the speed has dropped. Therefore, we created a new algorithm that solved the speed issue when validating the system on an embedded system.

With all the work we have developed and dedicated our time to, it requires us to do a lot of research about image preprocessing, deep learning, object detection, object tracking, and much more knowledge in the related field. Then, we proposed a solution for each task where the ability of each algorithm to answer our set objective they are improved from other research. Finally, we hope our research thesis would benefit and be useful to those who are interested in this research field and who have a motivation to enhance the technology in smart cities.





## REFERENCES

- Andrei, M.-A., Boianuiu, C.-A., Tarbă, N., & Voncilă, M.-L. (2022) Robust Lane Detection and Tracking Algorithm for Steering Assist Systems. doi: <https://doi.org/10.3390/machines10010010>
- Azhar, M. I. H., Zaman, F. H. K., Tahir, N. M., & Hashim, H. (2018). People tracking system using DeepSORT. *10th IEEE International Conference on Control System, Computing and Engineering (ICCSCCE)*. (pp. 137-141)
- Bangkok Biz News. (2017) More than 7,000 motorcycles get caught within 2 months for driving on the footpath and driving in the wrong direction. Retrieved from <https://www.bangkokbiznews.com/news/762121>.
- Bochkovskiy, A., Wang, C. Y., & Liao, H. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv Trans. Computer Vision and Pattern Recognition (cs.CV); Image and Video Processing (eess.IV)*. Academia Sinica, Taiwan.
- Canny, J. (1986) A Computational Approach To Edge Detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*. (8(6):679–698).
- Chen, L., Xu, X., Pan, L., Cao, J., & Li, X. (2021) Real-time lane detection model based on non-bottleneck skip residual connections and attention pyramids. *PLoS ONE* 16(10): e0252755. <https://doi.org/10.1371/journal.pone.0252755>.
- Duda, R.O., & Hart, P. E. (1972) Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Comm. ACM*. (Vol. 15, pp. 11–15).
- Farag, W., & Saleh, Z. (2018) Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems. *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. (pp. 1-8). doi: 10.1109/3ICT.2018.8855797.
- Franco, F., Santos, M.M.D., Yoshino, R.T., Yoshioka, L.R., & Justo, J.F. (2021) ROADLANE—The Modular Framework to Support Recognition Algorithms of Road Lane Markings. *Appl. Sci.* 2021. doi: 10.3390/app112210783
- Gedraite, E., & Hadad, M. (2011) Investigation on the effect of a Gaussian Blur in image filtering and segmentation.

- Ghazali, K., Xiao, R., & Ma, J. (2012) Road Lane Detection Using H-Maxima and Improved Hough Transform. *2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation*. (pp. 205-208). doi: 10.1109/CIMSim.2012.31.
- Gunjal, P. R., Gunjal, B. R., Shinde, H. A., Vanam, S. M., & Aher, S. S. (2018). Moving Object Tracking Using Kalman Filter. *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*. (pp. 544-547). doi: 10.1109/ICACCT.2018.8529402.
- Gu, H., Ge. Z., Cao, E., Chen, M., Wei, T., Fu, X., & Hu. S. (2021) A Collaborative and Sustainable Edge-Cloud Architecture for Object Tracking with Convolutional Siamese Networks. *IEEE Transactions on Sustainable Computing*. (vol. 6, no. 1, pp. 144-154). doi: 10.1109/TSUSC.2019.2955317.
- HSL and HSV. Retrieved from: [https://en.wikipedia.org/wiki/HSL\\_and\\_HSV](https://en.wikipedia.org/wiki/HSL_and_HSV)
- ImageStat Module. Retrieved from <https://pillow.readthedocs.io/en/stable/reference/ImageStat.html>
- Jetson Nano Developer Kit. Retrieved from <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>
- Liu, L., Chen, X., Zhu, S., & Tan, P. (2021) CondLaneNet: a Top-to-down Lane Detection Framework Based on Conditional Convolution. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. (pp. 3773-3782).
- Mandal, V., & Adu-Gyamfi, Y. (2020). Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis.
- Monteiro, G., Ribeiro, M., & Marcos, J. (2007). Wrongway drivers detection based on optical flow. *2007 IEEE International Conference on Image Processing* (vol. 5 pp. V-141-V-144).
- Oltean, G., Florea, C., Orghidan, R., & Oltean, V. (2019). Towards real time vehicle counting using YOLO-tiny and fast motion estimation. *IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME)*. (pp. 240-243).
- OpenCV Bitwise AND, OR, XOR, and NOT. Retrieved from <https://pyimagesearch.com/2021/01/19/opencv-bitwise-and-or-xor-and-not/>

- Patel, O., Maravi, Y., Sharma, S. (2013) A Comparative Study of Histogram Equalization Based Image Enhancement Techniques for Brightness Preservation and Contrast Enhancement. *Signal & Image Processing: An International Journal*. 4. 10.5121/sipij.2013.4502.
- Pudasaini, D., & Abhari, A. (2020) Scalable Object Detection, Tracking and Pattern Recognition Model Using Edge Computing. *Spring Simulation Conference (SpringSim)* (pp. 1-11). doi: 10.22360/SpringSim.2020.CNS.003.
- Rahman, Z., Ami, A. M., & Ullah, M. A. (2020). A real-time wrong-way vehicle detection based on YOLO and centroid tracking. *2020 IEEE Region 10 Symposium (TENSYP)* (pp. 916–920).
- Rakotondrajao, F. & Jangsamsi, K. (2019) Road Boundary Detection for Straight Lane Lines Using Automatic Inverse Perspective Mapping. *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*. (pp. 1-2). doi: 10.1109/ISPACS48206.2019.8986330.
- Roszyk, K., Nowicki, M.R., & Skrzypczyński, P. (2022) Adopting the YOLOv4 Architecture for Low-Latency Multispectral Pedestrian Detection in Autonomous Driving. *Sensors* 2022, doi: 10.3390/s22031082.
- Sharif, M. (2011) A new approach to compute convex hull. *Innovative Systems Design and Engineering*. 2. 187-193.
- Sharma, A., Kumar, M., Gupta, R.K., & Kumar, R. (2021) Lane detection using Python. *IJRMPS* 2021, 9, 917.
- Song, H., Liang, H., Li, H., Dai, Z., & Yun, X. (2019) Vision-based vehicle detection and counting system using deep learning in highway scenes. *European Transport Research Review*. 11. 10.1186/s12544-019-0390-4.
- Suttiponpisarn, P., Charnsripinyo, C., Usanavasin, S., & Nakahara, H. (2021) Detection of Wrong Direction Vehicles on Two-Way Traffic. *the 13th International Conference on Knowledge and Systems Engineering*.
- The Shapely User Manual. Retrieved from <https://shapely.readthedocs.io/en/stable/manual.html>
- Thongphat, N. (2019) Motorcycles key to solving road deaths. Retrieved from <https://www.bangkokpost.com/opinion/opinion/1628238/motorbikes-key-to-solving-road-d>

- Usmankhujaev, D., Baydadaev, S., & Kwon, J. W. (2020). Real-Time, Deep Learning Based Wrong Direction Detection. Inha University, Korea.
- Wojke, N., Bewley, A., & Paulus, D. (2017) Simple Online and Realtime Tracking with a Deep Association Metric. doi: 10.48550/arXiv.1703.07402.
- Yadav, G., Maheshwari, S., & Agarwal, A. (2014) Contrast limited adaptive histogram equalization based enhancement for real time video system. *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 2392-2397). doi: 10.1109/ICACCI.2014.6968381.
- Yang, Y. (2020) FastMOT: High-Performance Multiple Object Tracking Based on Deep SORT and KLT. Retrieved from <https://github.com/GeekAlexis/FastMOT>.
- Zualkernan, I., Dhou, S., Judas, J., Sajun, A.R., Gomez, B.R., & Hussain, L.A. (2022) An IoT System Using Deep Learning to Classify Camera Trap Images on the Edge. *Computers 2022*. <https://doi.org/10.3390/computers11010013>.

## BIOGRAPHY

Name Pintusorn Suttiponpisarn  
Education 2020: Bachelor of Engineering (Computer Engineering)  
Sirindhorn International Institute of Technology  
Thammasat University

### Publications

Suttiponpisarn, P., Charnsripinyo, C., Usanavasin, S., & Nakahara, H. (2021) Detection of Wrong Direction Vehicles on Two-Way Traffic. *the 13th International Conference on Knowledge and Systems Engineering*.

Suttiponpisarn, P., Charnsripinyo, C., Usanavasin, S., & Nakahara, H. (2022) An Enhanced System for Wrong Way Driving Vehicle Detection with Road Boundary Detection Algorithm. *International Conference on Industry Science and Computer Sciences Innovation 2022*.